Logistic Model Trees†

Niels Landwehr
Institute for Computer Science, University of Freiburg, Freiburg, Germany.
landwehr@informatik.uni-freiburg.de

Mark Hall and Eibe Frank
Department of Computer Science, University of Waikato, Hamilton, New Zealand.
{mhall, eibe}@cs.waikato.ac.nz

June 10, 2004

Abstract. Tree induction methods and linear models are popular techniques for supervised learning tasks, both for the prediction of nominal classes and numeric values. For predicting numeric quantities, there has been work on combining these two schemes into 'model trees', i.e. trees that contain linear regression functions at the leaves. In this paper, we present an algorithm that adapts this idea for classification problems, using logistic regression instead of linear regression. We use a stagewise fitting process to construct the logistic regression models that can select relevant attributes in the data in a natural way, and show how this approach can be used to build the logistic regression models at the leaves by incrementally refining those constructed at higher levels in the tree. We compare the performance of our algorithm to several other state-of-the-art learning schemes on 36 benchmark UCI datasets, and show that it produces accurate and compact classifiers.

Keywords: Model tree, logistic regression, classification

† This is an extended version of a paper that appeared in the Proceedings of the 14th European Conference on Machine Learning (Landwehr et al., 2003).

1. Introduction

Two popular methods for classification are linear logistic regression and tree induction, which have somewhat complementary advantages and disadvantages. The former fits a simple (linear) model to the data, and the process of model fitting is quite stable, resulting in low variance but potentially high bias. The latter, on the other hand, exhibits low bias but often high variance: it searches a less restricted space of models, allowing it to capture nonlinear patterns in the data, but making it less stable and prone to overfitting. So it is not surprising that neither of the two methods is superior in general—earlier studies (Perlich et al., 2003) have shown that their relative performance depends on the size and the characteristics of the dataset (e.g., the signal-to-noise ratio).

It is a natural idea to try and combine these two methods into learners that rely on simple regression models if only little and/or noisy data is available and add a more complex tree structure if there is enough data to warrant such structure. For the case of predicting a numeric variable, this has lead to 'model trees', which are decision trees with linear regression models at the leaves. These have been shown to produce good results (Quinlan, 1992). Although it is possible to use model trees for classification tasks by transforming the classification problem into a regression task by binarizing the class (Frank et al., 1998), this approach produces several trees (one per class) and thus makes the final model harder to interpret.

A more natural way to deal with classification tasks is to use a combination of a tree structure and logistic regression models resulting in a single tree. Another advantage of using logistic regression is that explicit class probability estimates are produced rather than just a classification. In this paper, we present a method, called LMT (Logistic Model Trees), that follows this idea. We discuss a new scheme for selecting the attributes to be included in the logistic regression models, and introduce a way of building the logistic models at the leaves by refining logistic models that have been trained at higher levels in the tree, i.e. on larger subsets of the training data.

We evaluate the performance of LMT on 36 datasets taken from the UCI repository (Blake and Merz, 1998). Included in the experiments are the standard decision tree learners C4.5 (Quinlan, 1993) and CART (Breiman et al., 1984), linear logistic regression, and other tree-based classifiers, such as boosted C4.5, model trees fit to the class indicator variables (Frank et al., 1998), functional trees (Gama, 2004), naive Bayes trees (Kohavi, 1996), and a different algorithm for building logistic model trees: Lotus (Chan and Loh, 2004). The experiments show that LMT produces more accurate classifiers than C4.5, CART, logistic regression, model trees, functional trees, naive Bayes trees and Lotus. It is competitive with boosted decision trees, which are considered to be one of the best 'off the shelf' classification systems, while producing models that are easier to interpret. We also present empirical evidence that LMT smoothly adapts the tree size to the complexity of the data set.

The rest of the paper is organized as follows. In Section 2 we briefly discuss the two learning methods that LMT is based on: tree induction and logistic regression. Section 3 discusses related work on tree-based learning. In Section 4 we present the LMT algorithm for learning logistic model trees. Section 5 describes our experimental study, followed by a discussion of results. Finally, we draw some conclusions in Section 6.

2. Tree Induction and Logistic Regression

This section discusses the two basic approaches to learning that our method is based upon: tree induction and logistic regression. We briefly introduce the process of tree induction, discuss the application of regression to classification tasks, and then describe our implementation of logistic regression.

2.1. Tree Induction

The goal of supervised learning is to find a subdivision of the instance space into regions labeled with one of the target classes. Top-down tree induction finds this subdivision by recursively splitting the instance space, stopping when the regions of the subdivision are reasonably 'pure' in the sense that they contain examples with mostly identical class labels. The regions are labeled with the majority class of the examples in that region.

Important advantages of tree models (with axis-parallel splits) are that they can be constructed efficiently and are easy to interpret. A path in a decision tree basically corresponds to a conjunction of boolean expression of the form 'attribute = value' (for nominal attributes) or 'attribute $\leq$ value' (for numeric attributes), so a tree can be seen as a set of rules that say how to classify instances.

The goal of tree induction is to find a subdivision that is fine enough to capture the structure in the underlying domain but does not fit random patterns in the training data.

As an example, Figure 1 shows a sample of 500 instances from an artificial domain, namely the sign-boundary of the function

$$f(x_1, x_2) = x_1^2 + x_1 + x_2 + e,$$

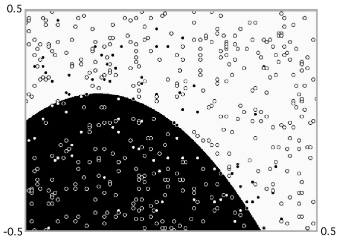

Figure 1. The artificial 'polynomial-noise' dataset and the uncorrupted class boundary.

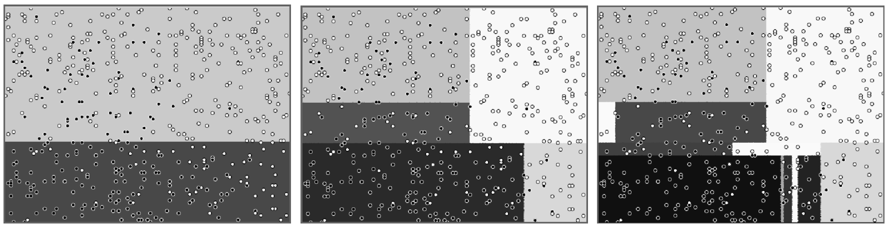

Figure 2. Subdivisions of increasing complexity for the 'polynomial-noise' dataset, generated by (from left to right) a decision stump learner, C4.5 with the 'minimum instances' parameter set to 20, and C4.5 with standard options. Colors (from light to dark) indicate class probability estimates in the different regions.

a polynomial of the two attributes x_1, x_2 that is corrupted by Gaussian noise e. The function was uniformly sampled in $[-1, 1]^2$. The original decision boundary of the polynomial (without noise) is also given (black/white region). We refer to this dataset as the 'polynomial-noise' dataset, it will be used again later.

Figure 2 shows three subdivision of the $\mathbf{R}^2$ instance space for the 'polynomial-noise' dataset, generated by a decision stump learner (i.e. a one-level decision tree), C4.5 (Quinlan, 1993) with the 'minimum instances' parameter set to 20, and C4.5 with standard options. They are increasingly more complex; in this case, the center one would probably be adequate, while the rightmost one clearly overfits the examples.

The usual approach to the problem of finding the best number of splits is to first perform many splits (build a large tree) and afterwards use a 'pruning' scheme to undo some of these splits. Different pruning schemes have been proposed. For example, C4.5 uses a statistically motivated estimate for the true error given the error on the training data, while the CART (Breiman et al., 1984) method cross-validates a 'cost-complexity' parameter that assigns a penalty to large trees.

2.2. Classification via Regression

The term 'regression' sometimes refers to a particular kind of parametric model for estimating a numeric target variable, and sometimes to the process of estimating a numeric target variable in general (as opposed to a discrete one). For the moment, we take the latter meaning—we explain how to solve a classification problem with a learner that can only produce estimates for a numeric target variable.

Assume we have a class variable G that takes on values $1, \dots, J$. The idea is to transform this class variable into J numeric 'indicator' variables $G_1, \dots, G_J$ to which the regression learner can be fit. The indicator variable G_j for class j takes on value 1 whenever class j is present and value 0 everywhere else. A separate model is then fit to every indicator variable G_j using the regression learner. When classifying an unseen instance, predictions $u_1, \dots, u_J$ are obtained from the numeric estimators fit to the class indicator variables, and the predicted class is

$$j^* = \operatorname*{argmax}_j \; u_j.$$

We will use this transformation process several times, for example when using model trees for classification.

Transforming a classification task into a regression problem in this fashion, we can use standard linear regression model for classification. Linear regression fits a parameter vector β to a numeric target variable to form a model

$$f(x) = \beta^T x$$

where x is the vector of attribute values for the instance (we assume a constant component in the input vector to accommodate the intercept). The model is fit to minimize the squared error:

$$\beta^* = \operatorname*{argmin}_\beta \; \sum_{i=1}^{n} (f(x_i) - y_i)^2,$$

where we have n training instances x_i that have target values y_i. However, this approach has some disadvantages. Usually, the predictions given by the regression functions fit to the class indicator variables are not confined to $[0, 1]$ and can even become negative. Besides, the approach is known to suffer from masking problems in the multiclass case: even if the class regions of the instance space are linearly separable, two classes can 'mask' a third one such that the learned model cannot separate it from the other two—see for example (Hastie et al., 2001).

2.3. Logistic Regression

A better way to use regression for classification tasks is to use a *logistic regression model* that models the posterior class probabilities $Pr(G = j|X = x)$ for the J classes. Given estimates for the class probabilities, we can classify unseen instances by

$$j^* = \underset{j}{\mathrm{argmax}}\ Pr(G = j|X = x).$$

Logistic regression models these probabilities using linear functions in x while at the same time ensuring they sum to one and remain in [0,1]. The model is specified in terms of $J-1$ log-odds that separate each class from the 'base class' J:

$$log\frac{Pr(G = j|X = x)}{Pr(G = J|X = x)} = \beta_j^T x, \quad j = 1, \ldots, J-1$$

or, equivalently,

$$Pr(G = j|X = x) = \frac{e^{\beta_j^T x}}{1 + \sum_{l=1}^{J-1} e^{\beta_l^T x}}, \quad j = 1, \ldots, J-1$$
$$Pr(G = J|X = x) = \frac{1}{1 + \sum_{l=1}^{J-1} e^{\beta_l^T x}}.$$

Note that this model still produces linear boundaries between the regions in the instance space corresponding to the different classes. For example, the x lying on the boundary between a class j and the class J are those for which $Pr(G = j|X = x) = Pr(G = J|X = x)$, which is equivalent to the log-odds being zero. Since the equation for the log-odds is linear in x, this class boundary is effectively a hyperplane. The formulation of the logistic model given here uses the last class as the base class in the odds-ratios; however, the choice of the base class is arbitrary in that the estimates are equivariant under this choice.

Fitting a logistic regression model means estimating the parameter vectors β_j. The standard procedure in statistics is to look for the *maximum likelihood* estimate: choose the parameters that maximize the probability of the observed data points. For the logistic regression model, there are no closed-form solutions for these estimates. Instead, we have to use numeric optimization algorithms that approach the maximum likelihood solution iteratively and reach it in the limit.

In a recent paper that links boosting algorithms like AdaBoost to additive modeling in statistics, Friedman et al. propose the LogitBoost algorithm for fitting *additive logistic regression models* by maximum

LogitBoost (J classes)

1. Start with weights $w_{ij} = 1/n,\ i = 1, \ldots, n,\ j = 1, \ldots, J,\ F_j(x) = 0$ and $p_j(x) = 1/J\ \ \forall j$
2. Repeat for $m = 1, \ldots, M$:
   (a) Repeat for $j = 1, \ldots, J$:
      i. Compute working responses and weights in the jth class
      $$z_{ij} = \frac{y^*_{ij} - p_j(x_i)}{p_j(x_i)(1 - p_j(x_i))}$$
      $$w_{ij} = p_j(x_i)(1 - p_j(x_i))$$
      ii. Fit the function $f_{mj}(x)$ by a weighted least-squares regression of z_{ij} to x_i with weights w_{ij}
   (b) Set $f_{mj}(x) \leftarrow \frac{J-1}{J}(f_{mj}(x) - \frac{1}{J}\sum_{k=1}^{J} f_{mk}(x)),\quad F_j(x) \leftarrow F_j(x) + f_{mj}(x)$
   (c) Update $p_j(x) = \frac{e^{F_j(x)}}{\sum_{k=1}^{J} e^{F_k(x)}}$
3. Output the classifier $\operatorname{argmax}_j\ F_j(x)$

Figure 3. LogitBoost algorithm (Friedman et al., 2000).

likelihood (Friedman et al., 2000). These models are a generalization of the (linear) logistic regression models described above. Generally, they have the form

$$Pr(G = j|X = x) = \frac{e^{F_j(x)}}{\sum_{k=1}^{J} e^{F_k(x)}}, \qquad \sum_{k=1}^{J} F_k(x) = 0,$$

where $F_j(x) = \sum_{m=1}^{M} f_{mj}(x)$ and the f_{mj} are (not necessarily linear) functions of the input variables. Indeed, the authors show that if regression trees are used as the f_{mj}, the resulting algorithm has strong connections to boosting decision trees with algorithms like AdaBoost.

Figure 3 gives the pseudocode for the algorithm. The variables y^*_{ij} encode the observed class membership probabilities for instance x_i, i.e.

$$y^*_{ij} = \begin{cases} 1 & \text{if } y_i = j, \\ 0 & \text{if } y_i \neq j \end{cases} \tag{1}$$

(recall that y_i is the class label of example x_i). The $p_j(x)$ are the estimates of the class probabilities for an instance x given by the model fit so far.

LogitBoost performs forward stagewise fitting: in every iteration, it computes 'response variables' z_{ij} that encode the error of the currently fit model on the training examples (in terms of probability estimates), and then tries to improve the model by adding a function f_{mj} to the committee F_j, fit to the response by least-squared error. As shown in (Friedman et al., 2000), this amounts to performing a quasi-Newton step in every iteration, where the Hessian matrix is approximated by its diagonal.

Any class of functions f_{mj} can be used as the 'weak learner' in the algorithm, as long as they are fit by a (weighted) least-squares regression. Depending on the class of functions, we get a more expressive or more restricted overall model. In the special case that the $f_{mj}(x)$ and so the $F_j(x)$ are linear functions of the input variables, the additive logistic regression model is equivalent to the linear logistic model introduced above. Assuming that $F_j(x) = \alpha_j^T x$, the equivalence of the two models is established by setting $\alpha_j = \beta_j - \beta_J$ for $j = 1 \dots J-1$ and $\alpha_J = \beta_J$. Note that the condition $\sum_{k=1}^{J} F_k(x) = 0$ is for stability only, adding a constant to all $F_k(x)$ does not change the model.

This means we can use the LogitBoost algorithm to learn linear logistic regression models, by fitting a standard least-squares regression function as the f_{mj} in step 2(a)ii. of the algorithm. In fact, in the two-class case this algorithm is equivalent to the standard 'iterative reweighted least squares' method used for fitting linear logistic regression models (Hastie et al., 2001).

2.3.1. *Attribute Selection*

Typical real-world data includes various attributes, only a few of which are actually relevant to the true target concept. If non-relevant attributes are included in, for example, a logistic regression model, they will usually allow the training data to be fitted with a smaller error, because there is by chance some correlation between the class labels and the values of these attributes for the training data. They will not, however, increase predictive power over unseen cases, and can sometimes even significantly reduce accuracy. Furthermore, including attributes that are not relevant will make it harder to understand the structure of the domain by looking at the final model, because it is 'distorted' by the influence of these attributes. Therefore, it is important to find some way to select the most relevant attributes to include in the logistic regression models.

When we say that we fit a linear regression function f_{mj} by least squares regression in a LogitBoost iteration, we may consider a *multiple* linear regression that makes use of all the attributes. However, it is also possible to use even simpler functions for the f_{mj}: simple regression functions, that perform a regression on only one attribute present in the training data. Fitting simple regression by least-squared error means fitting a simple regression function to each attribute in the data using least-squares as the error criterion, and then selecting the attribute that gives the smallest squared error.

Because every multiple linear regression can be expressed as a sum of simple linear regression functions, the general model does not change if we use simple instead of multiple regression for the f_{mj}. Furthermore, the final model found by LogitBoost will be the same because quasi-Newton stepping is guaranteed to actually find the maximum likelihood solution if the likelihood function is convex, which it is for linear logistic regression. Using simple regression functions instead of multiple ones will basically slow down the learning process, building gradually more complex models that include more and more attributes. However, all this only holds provided the algorithm is run until convergence (i.e., until the likelihood does not change anymore between two successive iterations). If it is stopped before it converges to the maximum likelihood solution, using simple regression will result in automatic attribute selection, because the model will only include the most relevant attributes present in the data. The stopping criterion can be based on cross-validation: only perform more iterations (and include more attributes) if this actually improves predictive accuracy over unseen instances.

On the other hand, slowing down the model fitting process can lead to higher computational costs. Although fitting a simple regression is computationally more efficient than fitting a multiple regression model, it could be necessary to consider the same attribute multiple times if the overall model has changed because other attributes have been included. This means many iterations have to be performed before the algorithm converges to a reasonable model. The computational complexity of a simple linear regression on one attribute is $O(n)$, so one iteration of LogitBoost would take $O(n \cdot a)$ because we have to build a simple regression model on all attributes in order to find out which one is the best (where n denotes the number of training examples and a the number of attributes present in the data). The computational complexity for performing a multiple regression is $O(n \cdot a^2 + a^3)$.[1] The relative speed of the two methods depends on how many LogitBoost iterations are required when using simple regression functions, but it is reasonable to expect that using multiple regression does converge faster.

[1] We take the number of classes J as a constant here, otherwise there is another factor of J.

We decided to use simple regression functions in our implementation because that approach improved predictive accuracy and significantly reduced the number of attributes included in the final model for some datasets (see Section 5.3 for an empirical comparison of this method to building a 'full' logistic model on all attributes). Note that we used simple regression in both the logistic model tree algorithm LMT that builds logistic regression functions at the nodes of a decision tree (see Section 4) and the standalone logistic regression learner we use as a benchmark in our experimental evaluation.

We determine the optimum number of LogitBoost iterations by a five fold cross-validation: we split the data five times into training and test sets, run LogitBoost on every training set up to a maximum number of iterations (500) and log the classification error on the respective test set. Afterwards, we run LogitBoost again on all data using the number of iterations that gave the smallest error on the test set averaged over the five folds. We will refer to this implementation as SimpleLogistic.

2.3.2. *Handling Nominal Attributes and Missing Values*

In real-world domains important information is often carried by nominal attributes whose values are not necessarily ordered in any way and thus cannot be treated as numeric (for example, the make of a car in the 'autos' dataset from the UCI repository). However, the regression functions used in the LogitBoost algorithm can only be fit to numeric attributes, so we have to convert those attributes to numeric ones. We followed the standard approach for doing this: a nominal attribute with k values is converted into k numeric indicator attributes, where the l-th indicator attribute takes on value 1 whenever the original attribute takes on its l-th value and value 0 everywhere else. Note that a disadvantage of this approach is that it can lead to a high number of attributes presented to the logistic regression if the original attributes each have a high number of distinct values. It is well-known that a high dimensionality of the input data (in relation to the number of training examples) increases the danger of overfitting. On such datasets, attribute selection techniques like the one implemented in SimpleLogistic will be particularly important.

Another problem with real-world datasets is that they often contain missing values, i.e. instances for which not all attribute values are observed. For example, an instance could describe a patient and attributes correspond to results of medical tests. For a particular patient results might only be available for a subset of all tests. Missing values can occur both during training and when predicting the class of an unseen instance. The regression functions that have to be fit in an iteration of

LogitBoost cannot directly handle missing values, so one has to fill in the missing values for such instances.

We used a simple global scheme for this: at training time, we calculate the mean (for numeric attributes) or the mode (for nominal attributes) of the values for each attribute and use these to replace missing values in the training data. When classifying unseen instances with missing values, the same mean/mode is used to fill in the missing value.

3. Related Tree-Based Learning Schemes

Starting from simple decision trees, several advanced tree-based learning schemes have been developed. In this section we will describe some of the methods related to logistic model trees, to show what our work builds on and where we improve on previous solutions. Some of the related methods will also be used as benchmarks in our experimental study, described in Section 5.

3.1. Model Trees

This section describes the 'model tree' algorithm developed by Quinlan, which combines regression and tree induction for tasks where the target variable to be predicted is numeric (Quinlan, 1992). The logistic model trees developed in this paper are an analogue to model trees for categorical target variables, so a description of model trees is a good starting point for understanding our method.

Model trees, like ordinary regression trees, predict a numeric value for an instance that is defined over a fixed set of numeric or nominal attributes. Unlike ordinary regression trees, model trees construct a piecewise linear (instead of a piecewise constant) approximation to the target function. The final model tree consists of a tree with linear regression functions at the leaves (Frank et al., 1998), and the prediction for an instance is obtained by sorting it down to a leaf and using the prediction of the linear model associated with that leaf.

The M5' model tree algorithm (Wang and Witten, 1997), which is a 'rational reconstruction' of Quinlan's M5 algorithm (Quinlan, 1992), constructs trees as follows. First, after all nominal attributes have been replaced by binary ones, an unpruned regression tree is grown, using variance reduction as the splitting criterion. Then, linear regression models are placed at every node of the tree, where the attributes considered in the regression are restricted to those that occur in the subtree rooted at the corresponding node. Further attribute selection in the linear models is performed by greedily dropping terms to minimize an error estimate that introduces a penalty for every parameter used in the model. Once all linear models are in place, subtrees are considered for replacement based on the final error estimate for each linear model.

At prediction time, the algorithm generates a 'smoothed' output by averaging the prediction of the linear model at a leaf node with the predictions obtained from the models on the path from that leaf to the root. The smoothing heuristic effectively performs a linear combination of linear models, which can be written as a linear model itself. Hence it is possible to achieve the same effect by replacing the original unsmoothed model at each leaf node with a smoothed version (Frank et al., 1998).

Model trees have been shown to produce good results for numeric prediction problems (Wang and Witten, 1997). They have also been successfully applied to classification problems using the transformation described in Section 2.2 (Frank et al., 1998). In our experimental section, we will give results for this 'M5' for classification' algorithm and compare it to our method.

3.2. Stepwise Model Tree Induction

In this section, we will briefly discuss a different algorithm for inducing (numeric) model trees called 'Stepwise Model Tree Induction' or SMOTI (Malerba et al., 2002) that builds on an earlier system called TSIR (Lubinsky, 1994). Although we are more concerned with classification problems, SMOTI uses a scheme for constructing the linear regression functions associated with the leaves of the model tree that is related to the way our method builds the logistic regression functions at the leaves of a logistic model tree. The idea is to construct the final multiple regression function at a leaf from simple regression functions that are fit at different levels in the tree, from the root down to that particular leaf. This means that the final regression function takes into account 'global' effects of some of the variables—effects that were not inferred from the examples at that leaf but from some superset of examples found on the path to the root of the tree. An advantage of this technique is that only simple linear regressions have to be fitted at the nodes of the tree, which is faster than fitting a multiple regression every time (that has to estimate the global influences again and again at the different nodes). The global effects should also smooth the predictions because there will be less extreme discontinuities between the linear functions at adjacent leaves if some of their coefficients have been estimated from the same (super)set of examples.

To implement these ideas, SMOTI trees consist of two types of nodes: split nodes and regression nodes. Split nodes partition the sample space in the usual way, while regression nodes perform simple linear regression on one attribute. A regression node fits a simple regression to the examples passed down to it from the parent node, and passes on a modified version of the examples to its only child node, removing the linear effect of the attribute used in the simple regression. This means the model at a leaf of the tree is constructed incrementally, adding more and more variables to it at the different regression nodes on the path to the leaf while the tree is grown. Our method uses a similar scheme for constructing the logistic regression models at the leaves: the simple regression functions produced in the iterations of the LogitBoost algorithm are fit on the nested sequence of sets of examples associated with the nodes on the path from the leaf to the root of the tree. Note, however, that our method is not restricted to a single simple linear model at each node. We will give a detailed description of this in Section 4.

3.3. Logistic Regression Trees with Unbiased Selection

Lotus (Chan and Loh, 2004) is a logistic regression tree learner for two class problems that has come from the statistics community. The algorithm constructs (binary) logistic regression trees in a top-down fashion, emphasizes the importance of unbiased split variable selection through the use of a modified chi-square test, and uses only numeric attributes for constructing logistic models. Lotus can fit either multiple or simple logistic regressions at the nodes.

After the initial tree is grown, it is pruned back using a pruning method similar to the one employed in the CART algorithm (Breiman et al., 1984). The idea is to use a 'cost-complexity measure' that combines the error of the tree on the training data with a penalty term for the model complexity, as measured by the number of terminal nodes. The cost-complexity-measure in CART is based on the misclassification error of a (sub)tree, whereas in Lotus it is based on the deviance. The deviance of a set of instances M is defined as

$$deviance = -2 \cdot logP(M|T)$$

where $P(M|T)$ denotes the probability of the data M as a function of the current model T (which is the tree being constructed).

3.4. Functional Trees

The LTree algorithm embodies a general framework for learning functional trees (Gama, 2004)—that is, multivariate classification or regression trees that can use combinations of attributes at decision nodes, leaf nodes, or both.

The algorithm uses a standard top-down recursive partitioning strategy to construct a decision tree. Splitting at each node is univariate, but considers both the original attributes in the data and new attributes constructed using an attribute constructor function: multiple linear regression in the regression setting and linear discriminants or multiple logistic regression in the classification setting. The value of each new attribute is the prediction of the constructor function for each example that reaches the node. In the classification case, one new attribute is created for each class and the values are predicted probabilities. In the regression case, a single new attribute is created. In this way the algorithm considers oblique splits based on combinations of attributes in addition to standard axis-parallel splits based on the original attributes. For split point selection, information gain is used in the classification case and variance reduction in the regression case.

Once a tree has been grown, it is pruned back using a bottom-up procedure. At each non-leaf node three possibilities are considered: performing no pruning (i.e, leaving the subtree rooted at the node in place), replacing the node with a leaf that predicts a constant, or replacing it with a leaf that predicts the value of the constructor function that was learned at the node during tree construction. C4.5's error-based criterion (Quinlan, 1993) is used to make the decision.

Predicting a test instance using a functional tree is accomplished by traversing the tree from the root to a leaf. At each decision node the local constructor function is used to extend the set of attributes and the decision test determines the path that the instance will follow. Once a leaf is reached, the instance is classifier using either the constant or the constructor function at that leaf (depending on what was put in place during the pruning procedure).

3.5. Naive Bayes Trees

NBTree is an algorithm that constructs decision trees with naive Bayes models at the leaves (Kohavi, 1996). A tree is grown in a top-down fashion with univariate splits chosen according to information gain. A pre-pruning strategy is employed during the construction of the tree that considers, at each node, whether the data at that node should be split, or a leaf created that contains a local naive Bayes model trained on the data at that node. The pruning decision at each node is made by comparing the cross-validated accuracy (computed using discretized data) of the local naive Bayes model at a node to the weighted sum of the accuracy of the leaves resulting from splitting at the node. A split is considered if there are at least 30 instances at a node and the relative reduction in error gained by splitting is greater than five percent.

It has been shown that naive Bayes trees often improve performance over standard decision trees or naive Bayes (Kohavi, 1996), although there are only a few cases where they improve significantly over both.

3.6. Boosting Trees

A well-known technique to improve the classification accuracy of tree-based classifiers is the boosting procedure. The idea of boosting is to combine the prediction of many 'weak' classifiers to form a powerful 'committee'. The weak classifiers are trained on reweighted versions of the training data, such that training instances that have been misclassified by the classifiers built so far receive a higher weight and the new classifier can concentrate on these 'hard' instances.

Although a variety of boosting algorithms have been developed, we will here concentrate on the popular AdaBoost.M1 algorithm (Freund and Schapire, 1996). The algorithm starts with equal weights assigned to all instances in the training set. One weak classifier (for, example, a C4.5 decision tree) is built and the data is reweighted such that correctly classified instances receive a lower weight: their weights are updated by

$$weight \leftarrow weight \cdot \frac{e}{1-e}$$

where e is the weighted error of the classifier on the current data. In a second step, the weights are renormalized such that their sum remains unchanged. This is repeated until the error e of a classifier reaches zero or exceeds 0.5 (or some pre-defined maximum for the number of boosting iterations is reached).

This procedure yields a set of classifiers with corresponding error values, which are used to predict the class of an unseen instance at classification time by a weighted majority vote. The vote of a classifier with error e is weighted by

$$\alpha = -log\frac{e}{1-e}.$$

For all classes, the weights of the classifiers that vote for it are summed up and the class with the largest sum of votes is chosen as the predicted class.

Boosting trees has received a lot of attention, and has been shown to outperform simple classification trees on many real-world domains. In fact, boosted decision trees are considered one of the best 'off-the-shelf' classifiers (learners that are not optimized with regard to a particular domain). On the other hand, boosted trees have some disadvantages compared to simple classification trees. One obvious disadvantage is the higher computational complexity, because the basic tree induction algorithm has to be run several times. But since basic tree induction is very fast, it is still feasible to build boosted models for most datasets. A more serious disadvantage is the reduced interpretability of a committee of trees as compared to a single tree. The interpretation of a tree as a set of rules does not translate to a whole set of trees which produce a classification by a weighted majority vote. However, information contained in the individual trees can still be used to yield some insight into the data, for example, the frequency of attributes occurring in the trees can tell us something about the relevance of that attribute for the class variable (see e.g. (Hastie et al., 2001)).

4. Logistic Model Trees

In this section, we will present our *Logistic Model Tree* algorithm, or LMT for short. It combines the logistic regression models described in Section 2 with tree induction, and thus is an analogue of model trees for classification problems.

4.1. The Model

A logistic model tree basically consists of a standard decision tree structure with logistic regression functions at the leaves, much like a model tree is a regression tree with regression functions at the leaves. As in ordinary decision trees, a test on one of the attributes is associated with every inner node. For a nominal (enumerated) attribute with k values, the node has k child nodes, and instances are sorted down one of the k branches depending on their value of the attribute. For numeric attributes, the node has two child nodes and the test consists of comparing the attribute value to a threshold: an instance is sorted down the left branch if its value for that attribute is smaller than the threshold and sorted down the right branch otherwise.

More formally, a logistic model tree consists of a tree structure that is made up of a set of inner or non-terminal nodes N and a set of leaves or terminal nodes T. Let S denote the whole instance space, spanned by all attributes that are present in the data. Then the tree structure gives a disjoint subdivision of S into regions S_t, and every region is represented by a leaf in the tree:

$$S = \bigcup_{t \in T} S_t, \qquad S_t \cap S_{t'} = \emptyset \;\; for \;\; t \neq t'$$

Unlike ordinary decision trees, the leaves $t \in T$ have an associated logistic regression function f_t instead of just a class label. The regression function f_t takes into account a subset $V_t \subseteq V$ of all attributes present in the data (where we assume that nominal attributes have been binarized for the purpose of regression), and models the class membership probabilities as

$$Pr(G = j|X = x) = \frac{e^{F_j(x)}}{\sum_{k=1}^{J} e^{F_k(x)}}$$

where

$$F_j(x) = \alpha_0^j + \sum_{v \in V_t} \alpha_v^j \cdot v,$$

or, equivalently,

$$F_j(x) = \alpha_0^j + \sum_{k=1}^{m} \alpha_{v_k}^j \cdot v_k$$

if $\alpha_{v_k}^j = 0$ for $v_k \notin V_t$. The model represented by the whole logistic model tree is then given by

$$f(x) = \sum_{t \in T} f_t(x) \cdot I(x \in S_t)$$

where $I(x \in S_t)$ is 1 if $x \in S_t$ and 0 otherwise.

Note that both standalone logistic regression and ordinary decision trees are special cases of logistic model trees, the first is a logistic model tree pruned back to the root, the second a tree in which $V_t = \emptyset$ for all $t \in T$.

Ideally, we want our algorithm to adapt to the dataset in question: for small datasets where a simple linear model offers the best bias-variance tradeoff, the logistic model 'tree' should just consist of a single logistic regression model, i.e. be pruned back to the root. For other datasets, a more elaborate tree structure is adequate.

The same reasoning also applies to the subsets of the original dataset that are encountered while building the tree. Recall that tree induction works in a divide-and-conquer fashion: a classifier for a set of examples is build by performing a split and then building separate classifiers for the two resulting subsets. There is strong evidence that building trees for very small datasets is usually *not* a good idea, it is better to use simpler models (like logistic regression) (Perlich et al., 2003). Because

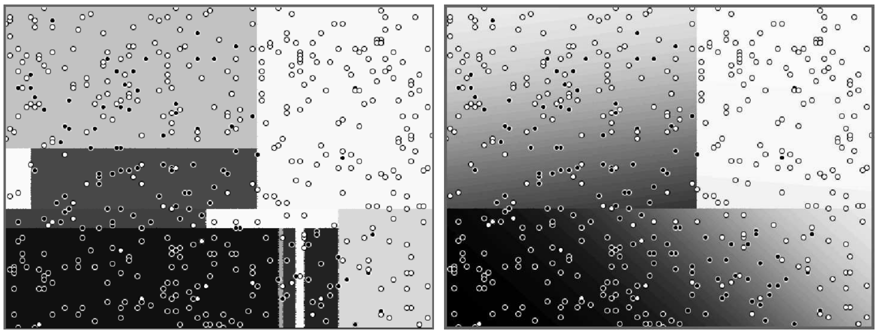

Figure 4. Class probability estimates from a C4.5 and a LMT model for the 'polynomial-noise' dataset. Colors (from light to dark) indicate class probability estimates in the different regions.

the subsets encountered at lower levels in the tree become smaller and smaller, it can be preferable at some point to build a linear logistic model instead of calling the tree growing procedure recursively. This is one motivation for the logistic model tree algorithm.

Figure 4 visualizes the class probability estimates of a logistic model tree and a C4.5 decision tree for the 'polynomial-noise' dataset introduced in Section 2.1. The logistic model tree initially divides the instance space into 3 regions and uses logistic regression functions to build the (sub)models within the regions, while the C4.5 tree partitions the instance space into 12 regions. It is evident that the tree built by C4.5 overfits some patterns in the training data, especially in the lower-right region of the instance space.

Figure 5 and Figure 6 depict the corresponding models. At the leaves of the logistic model tree, the functions F_1, F_2 determine the class membership probabilities by

$$Pr(G = 1|X = x) = \frac{e^{F_1(x)}}{e^{F_1(x)} + e^{F_2(x)}},$$

$$Pr(G = 2|X = x) = \frac{e^{F_2(x)}}{e^{F_1(x)} + e^{F_2(x)}}.$$

The entire left subtree of the root of the 'original' C4.5 tree has been replaced in the logistic model tree by the linear model with

$$\begin{aligned} F_1(x) &= -0.39 + 5.84 \cdot x_1 + 4.88 \cdot x_2 \\ F_2(x) &= 0.39 - 5.84 \cdot x_1 - 4.88 \cdot x_2 = -F_1(x) \end{aligned}$$

Note that this logistic regression function models a similar influence of the attributes x_1, x_2 on the class variable as the subtree it replaced, if we follow the respective paths in the tree we will see it mostly predicts

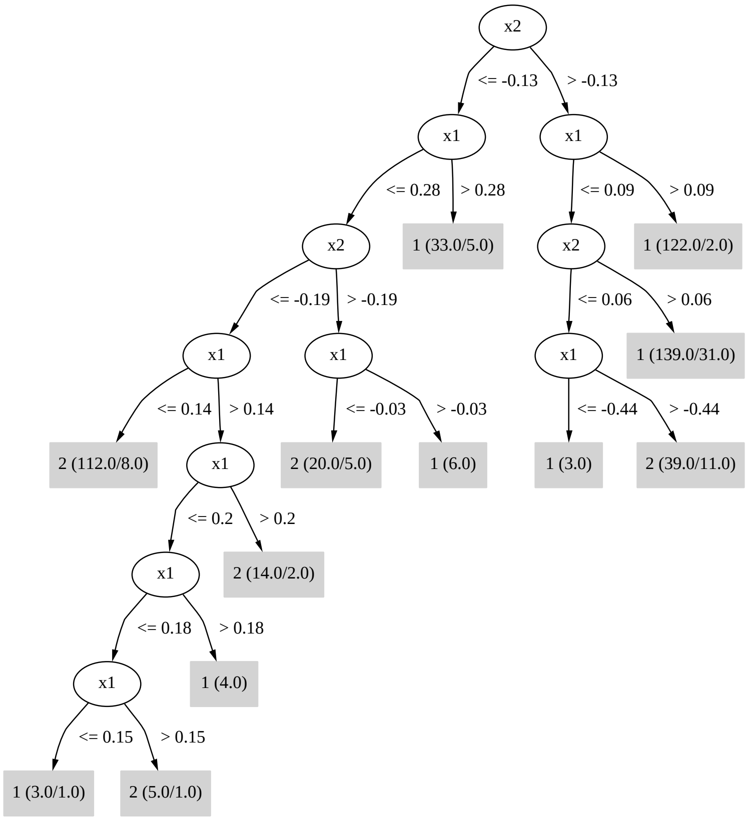

Figure 5. Decision tree constructed by the C4.5 algorithm for the 'polynomial-noise' dataset.

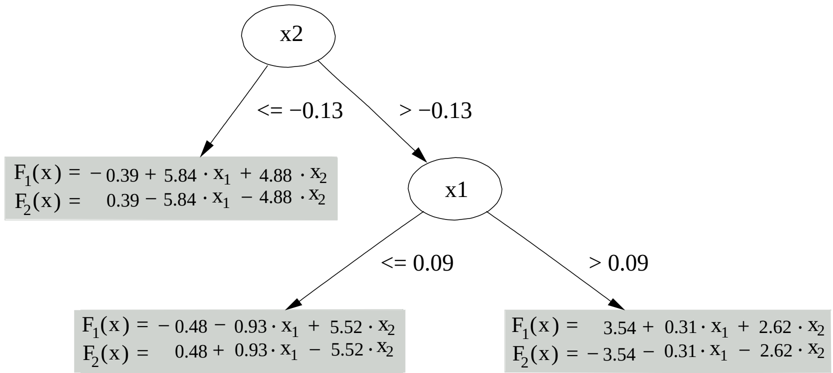

Figure 6. Logistic model tree constructed by the LMT algorithm for the 'polynomial-noise' dataset.

class one if x_1 and x_2 are both large. However, the linear model is simpler than the tree structure, and so less likely to overfit.

4.2. Building Logistic Model Trees

In the following we present our new algorithm for learning logistic model trees. We discuss how the initial tree is grown, the splitting and stopping criteria used, how the tree is pruned, and the handling of missing values and nominal attributes.

4.2.1. *Growing the Initial Tree*

There is a straightforward approach for growing logistic model trees that follows the way trees are built by M5'. This involves first building a standard classification tree, using, for example, the C4.5 algorithm, and afterwards building a logistic regression model at every node trained on the set of examples at that node (note that we need a logistic regression model at every node of the tree because every node is a 'candidate leaf' during pruning). In this approach, the logistic regression models are built in isolation on the local training examples at a node, not taking into account the surrounding tree structure.

Instead, we chose a different approach for constructing the logistic regression functions, namely by incrementally refining logistic models already fit at higher levels in the tree. Assume we have split a node and want to build the logistic regression function at one of the child nodes. Since we have already fit a logistic regression at the parent node, it is reasonable to use it as a basis for fitting the logistic regression at the child. We expect that the parameters of the model at the parent node already encode 'global' influences of some attributes on the class variable; at the child node, the model can be further refined by taking into account influences of attributes that are only valid locally, i.e. within the set of training examples associated with the child node.

The LogitBoost algorithm (discussed in Section 2.3) provides a natural way to do just that. Recall that it iteratively changes the linear class functions $F_j(x)$ to improve the fit to the data by adding a simple linear regression function f_{mj} to F_j, fit to the response variable. This means changing one of the coefficients in the linear function F_j or introducing a new variable/coefficient pair. After splitting a node we can continue running LogitBoost iterations, fitting the f_{mj} to the response variables of the training examples at the child node only.

As an example, consider a tree with a single split at the root and two successor nodes. The root node n has training data D_n and one of its children t has a subset of the training data $D_t \subset D_n$. Fitting the logistic regression models in isolation means the model f_n would

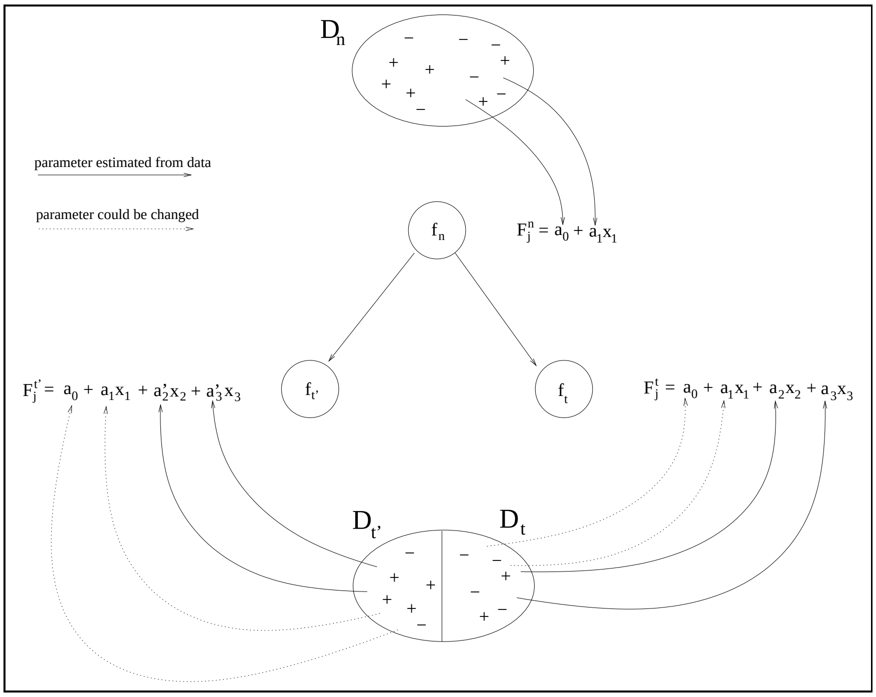

Figure 7. Building logistic models by incremental refinement. The parameters a_0, a_1 are estimated from the training examples at n, the parameters a_2, a_3 and a_2', a_3' from the training examples at t and t' respectively. Attribute x_1 has a global influence, x_2, x_3 have a local influence.

be built by iteratively fitting simple regression functions to D_n and the model f_t by iteratively fitting simple regression functions to D_t. In contrast, in the 'iterative refinement' approach, the tree is constructed as follows. We start by building a logistic model f_n at n by running LogitBoost on D_n, including more and more variables in the model by adding simple regressions f_{mj} to the F_j^n (the linear class function for each class j at node n). At some point, adding more variables does not increase the accuracy of the model,[2] but splitting the instance space and refining the logistic models locally in the two subdivisions created by the split might give a better model. So we split the node n and build refined logistic models at the child nodes by proceeding with the LogitBoost algorithm on the smaller set of examples D_t, adding more simple regression functions to the F_j^n to form the F_j^t. These simple linear regressions are fit to the response variables of the set of training

[2] This has to be determined using cross-validation or an independent test set, of course, because the training error will continue to decrease.

examples D_t given the (partial) logistic regression already fit at the parent node. Figure 7 illustrates this scheme for building the logistic regression models.

An additional advantage of this approach is that it is computationally more efficient to build the logistic models at lower levels of the tree by extending models already built at higher levels, rather than building the models at lower levels from scratch. Note that this approach of iteratively refining logistic regression models based on local structure in the data is related to the way SMOTI (discussed in Section 3.2) constructs linear models. In the terminology of the SMOTI system, our approach would amount to building a chain of 'regression nodes' as long as this improves the fit (as determined by the cross-validation), then a single 'split node' and again a chain of regression nodes.

These ideas lead to the following algorithm for building logistic model trees:

- Tree growing starts by building a logistic model at the root using the LogitBoost algorithm to iteratively fit simple linear regression functions, using five fold cross-validation to determine the appropriate number of iterations (i.e. using the SimpleLogistic algorithm described in Section 2.3).

- A split for the data at the root is constructed. Splits are either binary (for numeric attributes) or multiway (for nominal ones), the splitting criterion will be discussed in more detail below. Tree growing continues by sorting the appropriate subsets of data to the child nodes and building the logistic models at the child nodes in the following way: the LogitBoost algorithm is run on the subset associated with the child node, but starting with the committee $F_j(x)$, weights w_{ij} and probability estimates p_{ij} of the last iteration performed at the parent node (it is 'resumed' at step 2.a of Figure 3). Again, the optimum number of iterations to perform (the number of f_{jm} to add to F_j) is determined by a five fold cross validation.

- Splitting of the child nodes continues in this fashion until some stopping criterion is met (the stopping criterion is discussed in Section 4.2.2).

- Once the tree has been built it is pruned using CART-based pruning.

Figure 8 gives the pseudocode for this algorithm, which we call *LMT*. The method `LMT` constructs the tree given the training data

```
LMT(examples){
    root = new Node()
    alpha = getCARTAlpha(examples)
    root.buildTree(examples, null)
    root.CARTprune(alpha)
}

buildTree(examples, initialLinearModels) {
    numIterations =
       CV_Iterations(examples,initialLinearModels)
    initLogitBoost(initialLinearModels)
    linearModels = copyOf(initialLinearModels)
    for i = 1...numIterations
        logitBoostIteration(linearModels,examples)
    split = findSplit(examples)
    localExamples = split.splitExamples(examples)
    sons = new Nodes[split.numSubsets()]
    for s = 1...sons.length
        sons.buildTree(localExamples[s],nodeModels)
}

CV_Iterations(examples,initialLinearModels) {
    for fold = 1...5
        initLogitBoost(initialLinearModels)
        //split into training/test set
        train = trainCV(fold)
        test = testCV(fold)
        linearModels = copyOf(initialLinearModels)
        for i = 1...200
            logitBoostIteration(linearModels,train)
            logErrors[i] += error(test)
    numIterations = findBestIteration(logErrors)
    return numIterations
}
```

Figure 8. Pseudocode for the LMT algorithm.

`examples`. It calls `getCARTAlpha` to cross-validate the 'cost-complexity-parameter' for the CART pruning scheme implemented in `CARTPrune`.[3] The method `buildTree` grows the logistic model tree by recursively splitting the instance space. The argument `initialLinearModels` contains the simple linear regression functions already fit by LogitBoost at higher levels of the tree. The method `initLogitBoost` initializes the probabilities/weights for the LogitBoost algorithm as if it had already fitted the regression functions `initialLinearModels` (resuming LogitBoost at step 2.a). The method `CV_Iterations` determines the number of LogitBoost iterations to perform, and `logitBoostIteration` performs a single iteration of the LogitBoost algorithm (step 2), updating the probabilities/weights and adding a regression function to `linearModels`.

Some points in this sketch of the algorithm for growing logistic model trees need to be explained in more detail: how to select the attribute to split on, when to stop growing the tree, how to prune, and how to deal with missing values and nominal attributes.

4.2.2. *Splitting and Stopping*

We implemented two different criteria to select the attribute to split on. One is the C4.5 splitting criterion that tries to improve the purity of the class variable. The other splitting criterion attempts to divide the training data according to the current values of the working responses in the LogitBoost procedure in Figure 3.

In our experiments, we could not detect any significant differences between either classification accuracy or tree size between the two methods. A disadvantage of splitting on the response is that the final tree structure is less intelligible. Hence we made splitting on the class variable (using the C4.5 splitting criterion) the default option in our algorithm, and all experimental results reported for LMT in Section 5 refer to that version.

Tree growing stops for one of three reasons:

- A node is not split if it contains less than 15 examples. This number is somewhat larger than for standard decision trees, however, the leaves in logistic model trees contain more complex models, which need more examples for reliable model fitting.

- A particular split is only considered if there are at least 2 subsets that contain 2 examples each. Furthermore, a split is only considered if it achieves a minimum information gain. This is a heuristic used by the C4.5 algorithm to avoid overly fragmented splits. When no such split exists, we stop growing the tree.

[3] Note that this involves growing multiple 'auxiliary' logistic model trees.

– A logistic model is only built at a node if it contains at least 5 examples, because we need 5 examples for the cross-validation to determine the best number of iterations for the LogitBoost algorithm. Note that this can lead to 'partially expanded' nodes, where for some branches no additional iterations of LogitBoost are performed and so the model at the child is identical to the model of the parent.

We have found that the exact stopping criterion is not very important because the final tree (after pruning) is usually much smaller than the tree that is initially grown anyway.

4.2.3. *Pruning the Tree*

As for standard decision trees, pruning is an essential part of the LMT algorithm. Standard decision trees are usually grown to minimize the training error, which decreases monotonically as more and more splits are performed and the tree becomes larger and larger. Large trees, however, are complex models with many degrees of freedom, which means they can easily overfit random patterns in the training data that are not representative of the true structure of the domain.

The complexity of trees can be measured by the number of splits they contain: every new split further refines the subdivision of the instance space, and so makes the decision function represented by the tree more complex. Furthermore, splits usually introduce new parameters into the model (unless the same attribute has already been used in a different split). Pruning methods for decision trees make use of this fact by trading off tree size (which is roughly equivalent to the number of splits) versus training error.

For logistic model trees, the situation is slightly different compared to ordinary classification trees, because the logistic regression functions at the leaves are so much more complex than simple leaves (that just use the majority class in their set of examples for predictions). For logistic model trees, sometimes a single leaf (a tree pruned back to the root) leads to the best generalization performance, which is rarely the case for ordinary decision trees (there it would mean that the best thing to do is to predict the majority class for every unseen instance). So (correctly pruned) logistic model trees will usually be much smaller than ordinary classification trees, but the same principle still applies: every split and subsequent local refinement of the logistic regression models at the child nodes increases the complexity of the model. This means that pruning algorithms for decision trees that trade off tree size versus accuracy on the training set are still applicable. Note that if we had decided to build isolated logistic models at every node, it would not have been clear that a split really increases model complexity. Because the logistic models use variable selection, it could happen that a split plus two simple logistic models is actually less complex than a complex logistic model at the original node. This might lead to problems with a pruning algorithm that penalizes splits.

We spent a lot of time experimenting with different pruning schemes. Since our work was originally motivated by the model tree algorithm, we first tried adapting the pruning scheme used by this algorithm. However, we could not find a way to compute reliable estimates for the expected error rate (resulting in an unstable pruning algorithm), hence we abandoned that approach. Instead, we employed the pruning method from the CART algorithm (Breiman et al., 1984). Like M5's method, the CART pruning method uses a combination of training error and penalty term for model complexity to make pruning decisions. While estimating this parameter by cross-validation (or, if enough data is available, by using an independent test set) sacrifices some of the computational efficiency of M5's method, it leads to much more reliable pruning. The reader is referred to Breiman et al. (1984) for details of the pruning procedure.

4.2.4. *Handling of Missing Values and Nominal Attributes*

As explained in Section 2.3.2, missing values must be filled in before fitting the logistic regression models with the LogitBoost algorithm. This is done globally before tree building commences, by replacing the missing values with the means/modes of the respective attribute. This means that, unlike the C4.5 algorithm, LMT does not do any 'fractional splits' for instances with missing values (Quinlan, 1993). On the datasets we looked at, this simple approach seemed to work reasonably well, however, more sophisticated techniques could be an interesting area for future work.

Nominal attributes have to be converted to numeric ones in order to fit the logistic regression models. This is done locally at all nodes in our algorithm, i.e. the logistic model is fit to a local copy of the examples at a node where the nominal attributes have been transformed into binary ones. The procedure for this is the same as the one described in Section 2.3.2: a nominal attribute with k values is transformed into k binary indicator attributes that are then treated as numeric. The reason why this is not done globally is that splitting on nominal attributes (that often capture a specific characteristic of the domain) can be better than splitting on the binary ones that are the result of the conversion, both in terms of information gain and interpretability of the produced model.

4.3. Computational Complexity

This section discusses the computational complexity of building logistic regression models with SimpleLogistic and of building logistic model trees with LMT. It also describes two heuristics used to speed up the LMT algorithm.

Generally speaking, the stagewise model fitting approach used in SimpleLogistic means that a potentially large number of LogitBoost iterations have to be performed, because it might be necessary to fit a simple linear function to the same variable many times. The optimum number of iterations to be performed is selected by a five fold cross-validation. We used a maximum number of 200 iterations for the logistic regression at the nodes of the tree, as opposed to the 500 iterations used in stand-alone SimpleLogistic. The rationale for this is that the logistic regression functions in the tree do not have to be as complex as those for standalone logistic regression (because of the additional tree structure).

If the maximum number of iterations is taken as a constant, the asymptotic complexity of building a single logistic regression model is of the order $O(n \cdot a)$ (recall that n is the number of training examples and a the number of attributes). However, this ignores the constant factor of the number of LogitBoost iterations and the cross-validation. It is reasonable to expect that the maximum number of iterations should at least be linear in the number a of attributes present in the data (after all, the number of attributes that can be included in the final model is bounded by the number of LogitBoost iterations that can be performed). It is not clear whether our hard-coded limit is really enough for all datasets, though it seemed to work fine in our experiments. Therefore, a more realistic estimate of the asymptotic runtime of SimpleLogistic is $O(n \cdot a^2)$.

There is only a moderate increase in computational complexity from building logistic regression models to building logistic model trees. Using the more realistic estimate for the complexity of building the logistic models, the asymptotic complexity of the LMT algorithm is $O(d \cdot n \cdot \log n + n \cdot a^2 \cdot d + k^2)$, where d is the depth of the initial unpruned tree and k the number of nodes in the tree. The first part of the sum derives from building an unpruned decision tree, the second one from building the logistic regression models, and the third one from the CART pruning scheme. In our experiments, the time for building the logistic regression models accounted for most of the overall runtime. Note that the initial depth d of the unpruned logistic model tree is usually smaller than the depth of an unpruned standard classification tree, because tree growing is stopped earlier. The cross-validation performed by the CART pruning algorithm constitutes another constant multiplying factor of about six if five-fold cross-validation is used.

The asymptotic complexity is not too high compared to other machine learning methods, although it is higher than for simple tree induction. However, the two nested cross-validations—one for determining the optimum number of boosting iterations and one for the pruning—increase the runtime by a large constant factor, which makes the algorithm quite slow in practice.

4.3.1. *Heuristics to speed up the algorithm*

In this section we discuss two simple heuristics to speed up the LMT algorithm. The first one concerns the 'inner' cross-validation that determines the number of LogitBoost iterations to perform at a particular node. In order to avoid cross-validating this number at every node, we tried performing just *one* cross-validation to determine the optimum number of iterations for LogitBoost in the beginning (at the root of the tree) and then used that number *everywhere* in the tree. Although it is not very intuitive, this approach worked surprisingly well. It never produced results that were significantly worse than those of the original algorithm. It seems that the LMT algorithm is not too sensitive to the number of LogitBoost iterations that are performed at every node, as long as the number is roughly in the right range for the dataset. We also tried using some fixed number of iterations for every dataset, but that gave significantly worse results in some cases. It seems that the best number of iterations for LogitBoost does depend on the domain, but that it does not change so much for different subsets of a particular dataset (as encountered in lower levels in the tree).

As a second heuristic, we tried to stop performing LogitBoost iterations early in case it is obvious that the optimum number of iterations is relatively small. Recall that we run LogitBoost on the training set of a fold while monitoring the error on the test set, afterwards summing up the errors over the different folds to find the optimum number of iterations to perform. Examining the error curve on the test set produced by LogitBoost shows that the error usually decreases first, then reaches a minimum and later starts to increase again because the model is overfitting the training data. If the minimum is reached early, we can stop performing more iterations after a while because we know the best number must be relatively low. To account for spikes and irregularities in the error curve, we do not stop performing iterations immediately if the error increases, but instead keep track of the current minimum and stop if it has not changed for 50 iterations.

This second heuristic does not change the behavior of the algorithm significantly, and can give a considerable speed-up on datasets where

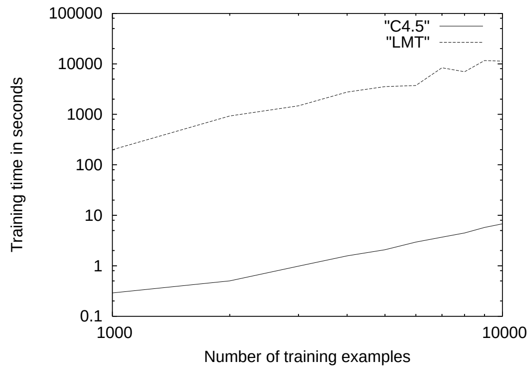

Figure 9. Training time as a function of the number of training instances for C4.5 and LMT on the letter dataset. Note the logarithmic scale of the axes.

the optimum number of LogitBoost iterations is small. Note that it can be used together with the first heuristic, by speeding up the initial cross-validation that determines the best number of LogitBoost iterations. By default, both heuristics are used in our implementation of LMT. All results shown for the LMT algorithm in this paper refer to the default version that uses the heuristics.

Figure 9 plots the training time as a function of training set size (note the log scales) of our implementations of LMT (including the heuristics) and C4.5 on the letter dataset from the UCI repository (Blake and Merz, 1998). The graph shows that LMT is several orders of magnitude slower than C4.5 in practice, although the shapes of their growth functions are comparable.

5. Experiments

In order to evaluate the performance of our method and compare it against other state-of-the-art learning schemes, we applied it to several real-world problems. More specifically, we seek to answer the following questions in our experimental study:

1. How does our version of logistic regression that uses parameter selection (SimpleLogistic) compare to a standard version of logistic regression that builds a full logistic model on all attributes present in the data?

2. How does LMT compare to the two algorithms that form its basis, i.e., logistic regression and C4.5? Ideally it should never perform worse than either of these algorithms.

3. How does LMT compare to other enhanced decision tree learners, i.e., those that make oblique splits or include other learning algorithms in the tree structure?

4. How does LMT compare to methods that build multiple trees? We include results for boosted C4.5 trees, where the final model is a 'voting committee' of trees, and for the M5' algorithm, building one tree per class.

5. How big are the trees constructed by LMT? We expect them to be much smaller than simple classification trees because the leaves contain more information. We also expect the trees to be pruned back to the root if a linear logistic model is the best solution for the dataset.

5.1. Algorithms Included in Experiments

The following algorithms are used in our experiments:[4]

– *C4.5*

  The C4.5 classification tree inducer (Quinlan, 1993). C4.5 is run with the standard options: The confidence threshold for pruning is 0.25, the minimum number of instances per leaf is 2. For pruning, both subtree replacement and subtree raising are considered.

– *CART*

  The CART tree inducer (Breiman et al., 1984). We used the implementation of CART provided in the R statistical package (Ihaka and Gentleman, 1996).[5] The cost-complexity parameter for pruning is optimized using 10-fold cross-validation on the training data.

– *LTree*

  The functional tree learning algorithm LTree (Gama, 2004). Linear discriminants or logistic regression can be used at interior nodes of the tree to provide oblique splits and at the leaves for prediction. We provide results for both types of discriminant functions. We used version 5.1 of LTree.[6]

[4] In the conference version of this paper (Landwehr et al., 2003) we also compared against the PLUS algorithm for inducing logistic model trees. We do not report results for PLUS here as the software is no longer available at time of writing and there is no publication describing it.

[5] R can be obtained from http://www.r-project.org.

[6] Available from http://www.liacc.up.pt/~jgama.

- *NBTree*

  The naive Bayes tree learning algorithm (Kohavi, 1996). NBTree is run using the settings recommended by Kohavi—a split is considered if there are at least 30 instances reaching a node and accepted if the relative reduction in error (computed by five-fold cross-validation) is greater than five percent.

- *M5'*

  The model tree learning algorithm M5' (Wang and Witten, 1997). The classification problems are transformed into regression problems as described in Section 2.2. M5' is run with the standard options: the minimum number of instances per leaf is four, smoothing is enabled.

- *AdaBoost.M1*

  The AdaBoost.M1 algorithm as described in Section 3.6. C4.5 with standard options is used as the base learner, and the maximum number of iterations is set to 10 or 100.

- *MultiLogistic*

  An implementation of logistic regression that finds a maximum-likelihood solution for a full logistic model. See Section 5.3 for more details.

- *SimpleLogistic*

  The standalone logistic regression with attribute selection as described in Section 2.3.

- *Lotus*

  The Lotus algorithm for inducing logistic regression trees with unbiased splits (Chan and Loh, 2004).[7] The algorithm can only handle two class datasets and can learn either simple or multiple logistic regression models at the leaves of the tree. We provide results for both cases. CART's cost-complexity pruning method is used with "cost" being predicted deviance estimated using 10-fold cross-validation on the training data. All other options (aside from logistic model type) are those computed as default by the Lotus software. We used version 1.03 of Lotus.

- *LMT*

  The LMT algorithm, using the heuristics discussed in Section 4.3.1.

[7] Lotus can be obtained from http://www.stat.nus.sg/~kinyee/lotus.html.

All algorithms except where noted are part of the Weka machine learning workbench (Witten and Frank, 2000) release 3.4.0.[8]

5.2. Datasets and Methodology

For the experiments we used the 36 benchmark datasets from the UCI repository (Blake and Merz, 1998) given in Table I. Their size ranges from under one hundred to a few thousand instances. They contain varying numbers of numeric and nominal attributes and some contain missing values. Note that the original zoo and splice datasets had an identifier attribute (that takes on a different value for every instance) which was removed for our experiments, because it leads to a sharp degradation in performance for M5' for classification.

For every dataset and algorithm, we performed ten runs of ten-fold stratified cross-validation (using the same splits into training/test set for every method). This gives a hundred data points for each algorithm and dataset, from which the average classification accuracy and standard deviation are calculated. Furthermore, we used a corrected resampled t-test (Nadeau and Bengio, 2003) instead of the standard t-test to identify if one method significantly outperforms another, at a 5 percent significance level. This test corrects for the dependencies in the estimates of the different data points, and is less prone to false-positive significance results. The statistic of the "corrected resampled t-test" is:

$$t = \frac{\frac{1}{N}\sum_{j=1}^{N} x_j}{\sqrt{(\frac{1}{N} + \frac{n_2}{n_1})\hat{\sigma}^2}}$$

where N is the number of data points (a hundred in our case), x_j is the difference in accuracy between two learning algorithms on one of the one hundred folds, n_1 is the number of instances used for training (ninety percent of the data in a ten-fold cross-validation) and n_2 is the number of test instances (ten percent of the data).

5.3. The Impact of Variable Selection for Logistic Regression

This section explores the effectiveness of our parameter selection method for SimpleLogistic, comparing it to MultiLogistic. There are two motivations for doing variable selection in logistic regression: one is the hope that controlling the number of parameters that enter the model can decrease the risk of building overly complex models that overfit the training data. This means that attribute selection should increase

[8] Weka is available from http://www.cs.waikato.ac.nz/˜ml/weka.

Table I. Datasets used for the experiments, sorted by size.

Dataset	Instances	Missing values (%)	Numeric attributes	Binary attributes	Nominal attributes	Classes
labor	57	35.7	8	3	5	2
zoo	101	0.0	1	15	0	7
lymphography	148	0.0	3	9	6	4
iris	150	0.0	4	0	0	3
hepatitis	155	5.6	6	13	0	2
glass (G2)	163	0.0	9	0	0	2
autos	205	1.1	15	4	6	6
sonar	208	0.0	60	0	0	2
glass	214	0.0	9	0	0	6
audiology	226	2.0	0	61	8	24
heart-statlog	270	0.0	13	0	0	2
breast-cancer	286	0.3	0	3	6	2
heart-h	294	20.4	6	3	4	2
heart-c	303	0.2	6	3	4	2
primary-tumor	339	3.9	0	14	3	21
ionosphere	351	0.0	33	1	0	2
horse-colic	368	23.8	7	2	13	2
vote	435	5.6	0	16	0	2
balance-scale	625	0.0	4	0	0	3
soybean	683	9.8	0	16	19	19
australian	690	0.6	6	4	5	2
breast-w	699	0.3	9	0	0	2
pima-indians	768	0.0	8	0	0	2
vehicle	846	0.0	18	0	0	4
anneal	898	0.0	6	14	18	5
vowel	990	0.0	10	2	1	11
german	1000	0.0	6	3	11	2
segment	2310	0.0	19	0	0	7
splice	3190	0.0	0	0	60	3
kr-vs-kp	3196	0.0	0	35	1	2
hypothyroid	3772	5.5	7	20	2	4
sick	3772	5.5	7	20	2	2
waveform	5000	0.0	40	0	0	3
optdigits	5620	0.0	64	0	0	10
pendigits	10992	0.0	16	0	0	10
letter	20000	0.0	16	0	0	26

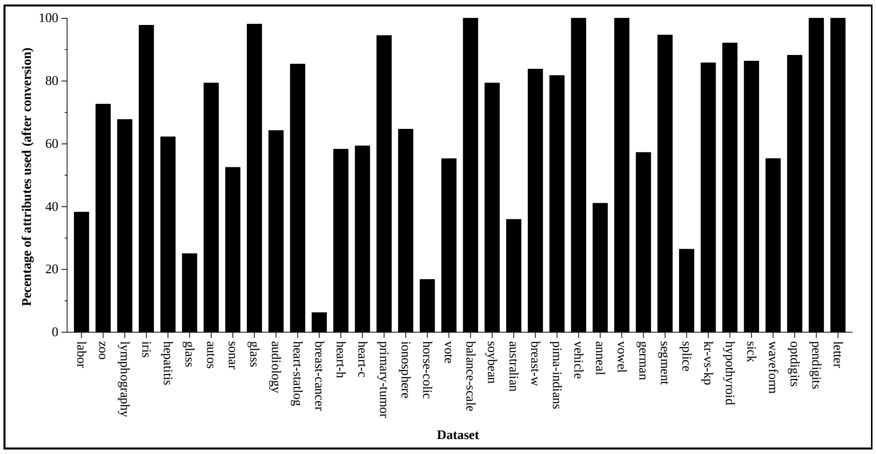

Figure 10. Percentage of attributes used by SimpleLogistic.

the classification accuracy. The other motivation is that models with many parameters are usually harder to interpret than models with few parameters; in particular, parameters that have no real relation to the target variable can be misleading when interpreting the final model.

Table II gives the classification accuracy for the two methods, and indicates significant wins/losses according to the modified t-test discussed above. The test reports four significant accuracy differences in favor of SimpleLogistic. Ignoring the significance of the differences, SimpleLogistic is better than MultiLogistic on 24 datasets and worse on 12 datasets. According to a two-tailed sign test, SimpleLogistic is more accurate than MulitLogistic at the 10 percent significance level (p-value is 0.0652). Hence attribute selection leads to better predictive performance for some datasets and never significantly decreases performance.

Figure 10 gives the percentage of attributes (after converting nominal attributes to binary ones) included in the final model for SimpleLogistic. Although the fraction of attributes that are discarded varies wildly (from more than 95 percent for the breast-cancer dataset to none for balance-scale, vehicle, vowel, pendigits and letter), in most cases the number of attributes included in the final model is reduced substantially. On average, the biggest reduction takes place for datasets with a high number of attributes that are not too large (datasets are sorted by size from left to right).

As an example for parameter selection, consider the breast-cancer dataset. After transforming the nominal attributes into binary ones, the dataset has 48 numeric attributes (plus the class). SimpleLogistic builds a model including only two of the parameters. The function

Table II. Mean classification accuracy and standard deviation for SimpleLogistic and MultiLogistic.

Dataset	SimpleLogistic	MultiLogistic
labor	91.97±10.54	94.07±10.01
zoo	94.69±6.59	94.85±6.34
lymphography	84.37±9.97	77.58±10.59
iris	95.93±4.82	97.07±4.77
hepatitis	84.20±8.20	83.89±8.12
glass (G2)	76.32±8.89	69.56±10.77
autos	75.30±9.97	69.99±9.86
sonar	75.93±8.51	72.47±8.90
glass	65.29±8.03	63.12±9.16
audiology	84.10±7.35	79.71±8.36
heart-statlog	83.30±6.48	83.67±6.43
breast-cancer	74.94±6.25	67.77±6.92 •
heart-h	84.61±6.03	84.23±5.93
heart-c	83.30±6.35	83.70±6.64
primary-tumor	47.87±6.04	41.56±7.63 •
ionosphere	87.78±4.99	87.72±5.57
horse-colic	81.93±5.80	80.87±6.06
vote	95.93±2.58	95.65±3.12
balance-scale	88.74±2.91	89.44±3.29
soybean	93.43±2.54	92.91±2.67
australian	85.04±3.97	85.33±3.85
breast-w	96.21±2.19	96.50±2.18
pima-indian	77.10±4.65	77.47±4.39
vehicle	80.45±3.37	79.81±4.04
anneal	99.48±0.70	99.24±0.79
vowel	84.31±3.78	82.71±3.96
german	75.34±3.50	75.24±3.54
segment	95.40±1.50	95.60±1.59
splice	95.86±1.17	90.57±1.85 •
kr-vs-kp	97.46±0.79	97.56±0.76
hypothyroid	96.78±0.73	96.69±0.72
sick	96.74±0.71	96.79±0.69
waveform	86.96±1.58	86.73±1.49
optdigits	97.12±0.68	93.89±0.98 •
pendigits	95.51±0.64	95.50±0.65
letter	77.42±0.80	77.40±0.83

•, ◦ statistically significant win or loss for SimpleLogistic determining the class probability membership (cf. Section 2.3) for class 1, no-recurrence-events, is

$$F_1(x) = 0.53 + [inv - nodes = 0 - 2] \cdot 1.07 - [deg - malig = 3] \cdot 1.32.$$

The function F_2 for the class membership probability of class 2 is $F_2(x) = -F_1(x)$ because there are only two classes. The binary attribute [inv-nodes=0–2] has been generated from the nominal attribute 'inv-nodes' that can take on values '0–2', '3–5', and so on. The nominal attribute 'deg-malig' takes on values '1', '2' and '3'. The model is relatively easy to interpret: it basically says that no recurrence events are expected if the number of involved nodes is small and the degree of malignancy is not too high.

In contrast to that, the model built by MultiLogistic has sizeable coefficients for 38 of the 48 attributes, which makes it a lot harder to interpret, and is at the same time less accurate (see Table II).

5.4. Empirical Evaluation of LMT

This section discusses the performance of LMT compared to other learning schemes, including logistic regression, C4.5, CART, LTree, Lotus, M5' for classification, and boosted C4.5 trees.

5.4.1. *Comparing LMT to Logistic Regression and Tree Induction*

Table III gives the average classification accuracy and standard deviation for LMT, SimpleLogistic, MultiLogistic, C4.5 and CART. Table IV shows how the different methods compare with each other. Each entry indicates the number of datasets for which the method associated with its column is (significantly) more accurate than the method associated with its row.

We observe that LMT indeed always reaches roughly the same classification accuracy as logistic regression and the other tree learners: there is no dataset where LMT is significantly outperformed by either SimpleLogistic, MultiLogistic, C4.5, or CART. It significantly outperforms SimpleLogistic on 8 datasets, MultiLogistic on 13 datasets, C4.5 on 16 datasets and CART on 17 datasets. We can also confirm the observation (see for example (Lim et al., 2000)) that logistic regression performs surprisingly well compared to tree induction on most UCI datasets. This includes all small to medium-sized datasets except ionosphere. Only on some larger datasets (kr-vs-kp, sick, hypothyroid, pendigits and letter) is its performance not competitive with that of tree induction (and other methods, see below).

Ignoring the significance of individual differences, a two-tailed sign test confirms that LMT outperforms the other methods. LMT has a

Table III. Mean classification accuracy and standard deviation for LMT vs. SimpleLogistic, MultiLogistic, C4.5 and CART.

Dataset	LMT	SimpleLogistic	MultiLogistic	C4.5	CART
labor	91.37±11.01	91.97±10.54	94.07±10.01	78.60±16.58 •	81.73±17.54
zoo	94.89±6.44	94.69±6.59	94.85±6.34	92.61±7.33	91.81±7.44
lymphography	84.10±10.00	84.37±9.97	77.58±10.59	75.84±11.05	76.86±9.87
iris	95.80±4.89	95.93±4.82	97.07±4.77	94.73±5.30	96.00±4.74
hepatitis	84.21±8.21	84.20±8.20	83.89±8.12	79.22±9.57	80.21±7.57
glass (G2)	77.28±9.90	76.32±8.89	69.56±10.77 •	78.15±8.50	80.21±9.51
autos	76.13±8.84	75.30±9.97	69.99±9.86	81.77±8.78	77.66±10.20
sonar	76.32±9.58	75.93±8.51	72.47±8.90	73.61±9.34	74.77±9.76
glass	69.15±8.99	65.29±8.03	63.12±9.16	67.63±9.31	68.09±10.49
audiology	84.19±7.17	84.10±7.35	79.71±8.36	77.26±7.47 •	77.01±7.75 •
heart-statlog	83.22±6.50	83.30±6.48	83.67±6.43	78.15±7.42 •	78.00±8.25 •
breast-cancer	74.91±6.29	74.94±6.25	67.77±6.92 •	74.28±6.05	69.40±5.25 •
heart-h	84.00±6.33	84.61±6.03	84.23±5.93	80.22±7.95	76.70±6.80 •
heart-c	83.51±6.67	83.30±6.35	83.70±6.64	76.94±6.59 •	78.88±6.87 •
primary-tumor	47.63±5.84	47.87±6.04	41.56±7.63 •	41.39±6.94 •	41.42±7.38 •
ionosphere	92.99±4.13	87.78±4.99 •	87.72±5.57 •	89.74±4.38 •	89.80±4.78
horse-colic	83.66±6.13	81.93±5.80	80.87±6.06	85.16±5.91	84.63±5.56
vote	95.90±2.67	95.93±2.58	95.65±3.12	96.57±2.56	95.15±3.08
balance-scale	89.71±2.68	88.74±2.91	89.44±3.29	77.82±3.42 •	78.09±3.97 •
soybean	93.43±2.59	93.43±2.54	92.91±2.67	91.78±3.19	90.80±3.15 •
australian	85.04±3.84	85.04±3.97	85.33±3.85	85.57±3.96	84.55±4.20
breast-w	96.18±2.20	96.21±2.19	96.50±2.18	95.01±2.73	94.42±2.70 •
pima-indians	77.08±4.65	77.10±4.65	77.47±4.39	74.49±5.27	74.50±4.70
vehicle	83.04±3.70	80.45±3.37	79.81±4.04 •	72.28±4.32 •	72.37±4.12 •
anneal	99.52±0.73	99.48±0.70	99.24±0.79	98.57±1.04 •	98.24±1.18 •
vowel	93.94±2.55	84.31±3.78 •	82.71±3.96 •	80.20±4.36 •	81.54±4.10 •
german	75.37±3.53	75.34±3.50	75.24±3.54	71.25±3.17 •	73.34±3.66
segment	96.28±1.36	95.40±1.50 •	95.60±1.59	96.77±1.29	96.54±1.23
splice	95.86±1.17	95.86±1.17	90.57±1.85 •	94.17±1.28 •	94.72±1.34 •
kr-vs-kp	99.64±0.35	97.46±0.79 •	97.56±0.76 •	99.44±0.37	99.41±0.39
hypothyroid	99.54±0.39	96.78±0.73 •	96.69±0.72 •	99.54±0.36	99.66±0.30
sick	98.95±0.60	96.74±0.71 •	96.79±0.69 •	98.72±0.55	98.89±0.57
waveform	86.96±1.58	86.96±1.58	86.73±1.49	75.25±1.90 •	76.97±1.56 •
optdigits	97.28±0.64	97.12±0.68	93.89±0.98 •	90.52±1.20 •	90.78±1.27 •
pendigits	98.56±0.32	95.51±0.64 •	95.50±0.65 •	96.54±0.60 •	96.37±0.62 •
letter	92.13±0.74	77.42±0.80 •	77.40±0.83 •	88.03±0.71 •	87.62±0.81 •

•, ∘ statistically significant win or loss for LMT

Table IV. Pairwise comparison of classification accuracy for LMT, SimpleLogistic, MultiLogistic, C4.5 and CART: number indicates how often method in column (significantly) outperforms method in row.

	LMT	SimpleLogistic	MultiLogistic	C4.5	CART
LMT	-	12 (0)	8 (0)	7 (0)	6 (0)
SimpleLogistic	23 (8)	-	12 (0)	13 (7)	12 (6)
MultiLogistic	28 (13)	24 (4)	-	16 (10)	14 (7)
C4.5	29 (16)	23 (13)	20 (8)	-	20 (1)
CART	30 (17)	24 (14)	22 (11)	16 (1)	-

win/loss-ratio of 23/12 against SimpleLogistic, 28/8 against MultiLogistic, 29/7 against C4.5 and 30/6 against CART. The ratio against SimpleLogistic is significant at the 10 percent level, while the others are significant at the one percent level (p-values of 0.0895, 0.0012, 0.0003 and < 0.0001 respectively).

Table V gives the average tree size and standard deviation for LMT, C4.5 and CART. Table VI summarizes the results. As expected, the trees constructed by LMT are much smaller than the standard classification trees built by C4.5. LMT produces significantly smaller trees on all datasets. Of course, logistic model trees contain part of their 'structure' in the leaves in form of the logistic regression functions. Nevertheless, there are some datasets where the difference in tree size is so large that one advantage of C4.5, namely that the final models are potentially easier to understand, disappears. For the waveform dataset, for example, LMT builds a logistic regression model (no tree structure), while C4.5 builds a tree with almost 300 terminal nodes. About half of the 40 attributes in waveform are used in the logistic regression (see Table 10). It is probably easier to understand the influence of the attributes on the class variable from a logistic model with 20 parameters than from a tree with 300 terminal nodes. Looking at the results for CART, we can see that the smaller trees produced by LMT are not just an artifact of the cost-complexity pruning. LMT produces significantly smaller trees than CART on all but four of the datasets. Although CART generally produces smaller trees than C4.5 (due to its pruning mechanism), the difference in size is still dramatic on many of the datasets. For instance, on the the letter dataset LMT builds a tree with 41 leaves, while CART (and C4.5) produces a tree with over a thousand leaves.

There are several datasets for which the trees constructed by LMT are pruned back to the root. Let us a call a tree "pruned back to the

Table V. Mean tree size (number of leaves) and standard deviation for LMT vs. C4.5 and CART.

Dataset	LMT	C4.5		CART	
labor	1.05±0.26	4.16±1.44	•	3.96±0.98	•
zoo	1.05±0.26	8.35±0.82	•	7.09±0.40	•
lymphography	1.18±0.80	17.30±2.88	•	9.84±4.56	•
iris	1.11±0.53	4.64±0.59	•	4.72±0.81	•
hepatitis	1.05±0.30	9.33±2.37	•	5.30±3.02	•
glass (G2)	5.05±3.24	12.53±2.64	•	6.70±2.86	
autos	2.89±5.20	44.77±6.09	•	19.31±1.91	•
sonar	2.12±1.51	14.45±1.75	•	9.49±3.09	•
glass	7.46±3.55	23.58±2.29	•	19.29±5.96	•
audiology	1.01±0.10	29.90±1.95	•	17.51±3.67	•
heart-statlog	1.04±0.32	17.82±2.86	•	8.97±4.43	•
breast-cancer	1.12±1.20	9.75±8.16	•	6.29±4.83	•
heart-h	1.00±0.00	6.32±3.73	•	8.22±6.93	•
heart-c	1.03±0.17	25.70±5.53	•	8.50±4.03	•
primary-tumor	1.03±0.17	43.81±5.16	•	19.51±13.55	•
ionosphere	4.40±1.86	13.87±1.95	•	6.34±3.80	
horse-colic	2.78±2.58	5.91±1.99	•	4.06±1.77	
vote	1.08±0.44	5.83±0.38	•	6.45±2.51	•
balance-scale	4.73±2.61	41.60±4.97	•	44.28±12.51	•
soybean	2.44±5.44	61.12±6.01	•	40.76±6.13	•
australian	1.15±0.59	22.49±7.54	•	6.48±5.33	•
breast-w	1.24±0.99	12.23±2.77	•	11.17±4.46	•
pima-indians	1.03±0.30	22.20±6.55	•	14.92±12.96	•
vehicle	2.64±1.97	69.50±10.28	•	61.43±20.93	•
anneal	1.11±0.31	37.98±5.39	•	12.59±1.99	•
vowel	4.81±1.23	123.28±17.19	•	87.17±4.27	•
german	1.00±0.00	90.18±15.67	•	16.58±12.90	•
segment	4.85±2.05	41.20±3.04	•	41.08±4.30	•
splice	1.00±0.00	174.39±12.21	•	25.82±9.03	•
kr-vs-kp	7.92±0.56	29.29±1.83	•	30.70±3.41	•
hypothyroid	5.61±0.93	14.44±1.06	•	6.16±0.42	
sick	13.56±3.12	27.44±3.88	•	20.63±3.45	•
waveform	1.00±0.00	296.49±12.16	•	87.88±29.68	•
optdigits	3.45±1.66	205.69±7.26	•	165.92±17.10	•
pendigits	11.30±2.78	188.20±5.45	•	182.65±13.12	•
letter	41.73±8.68	1160.92±16.79	•	1125.86±19.20	•

•, ∘ statistically significantly smaller or larger trees for LMT

Table VI. Pairwise comparison of tree size (number of leaves) for LMT, C4.5 and CART: number indicates how often method in column has (significantly) larger tree size than method in row.

	LMT	C4.5	CART
LMT	-	36 (36)	36 (32)
C4.5	0 (0)	-	5 (0)
CART	0 (0)	31 (23)	-

root" if the average tree size is less than 1.5, meaning that more than half of the data points correspond to 'trees' that were just a logistic regression function. It can be seen from Table V that this happens on exactly half of the 36 datasets. If the pruning process worked correctly, we would expect that for the 18 datasets where the tree is pruned back to the root the classification accuracy is better than that of the C4.5 algorithm—indicating that a linear logistic regression model is preferable to a tree structure—and for the other 18 datasets the result for LMT is better than that of SimpleLogistic. The first claim is true for all but two datasets (vote and australian, where there is a small win for C4.5). The second claim is true for 17 out of the 18 datasets where a tree structure is built; on the single exception ('soybean') the result for LMT is equal to that of SimpleLogistic. Allowing for small random effects, we can conclude that the adapted pruning method—the CART algorithm from (Breiman et al., 1984)—reliably makes the right choice between building a linear logistic model and a more elaborate tree structure.

To illustrate how the LMT algorithm scales model complexity depending on the information available for training, we ran it on increasingly larger training datasets sampled from an artificial domain and compared its result to the results for SimpleLogistic ('logistic model tree of size one') and C4.5 (standard classification tree). Consider the polynomial

$$f : \mathbf{R}^4 \rightarrow \mathbf{R}, \;\; f(x) = 2 \cdot x_1^2 + x_2 + x_3 + x_4$$

and the binary function

$$g : \mathbf{R}^4 \rightarrow \{1, -1\}, \;\; g(x) = sign(f(x)).$$

The function $g(x)$ gives rise to a two-class classification problem over the four numeric attributes $x_1, \ldots, x_4$. The attributes x_2, x_3, x_4 have

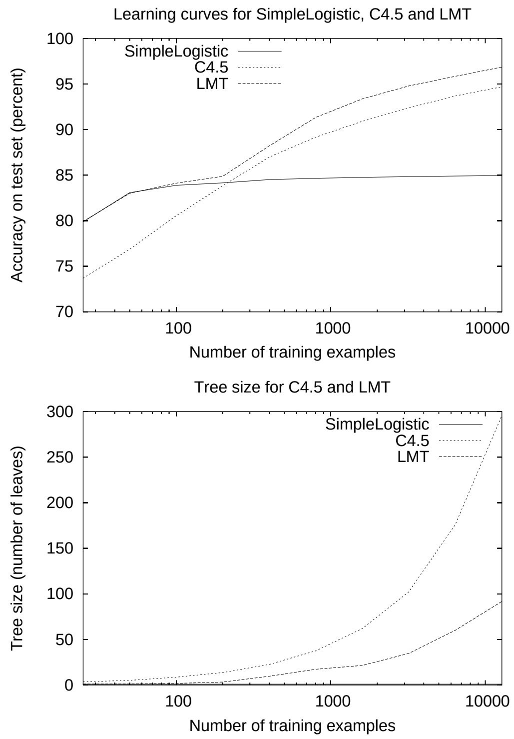

Figure 11. Accuracy (top) and tree size (bottom) as a function of the number of training instances. Note the log scale on the x axis.

a linear influence on the target variable while the influence of x_1 is nonlinear. We sampled training datasets of size 25, 50, 100, 200, 400, 800, 1600, 3200, 6400 and 12800 instances from this domain, then used LMT, SimpleLogistic and C4.5 to build a model on the training set and evaluated its performance on a separate test set. More specifically, we generated 100 datasets of each size, sampling $g(x)$ uniformly in $[-1,1]^4$. This gives a-priori probabilities for the two classes of about 0.7 for class 1 and 0.3 for class -1. Samples are stratified, meaning that the distribution of the classes in the sample is the same as their a-priori probability (as far as possible), which helps getting stable estimates for small training set sizes. The accuracy achieved by each method was measured using a fixed test set of 10000 instances. From the 100 data points measured for each algorithm and training set size, we calculated the average accuracy on the test set for every method and the average tree size for LMT and C4.5.

Figure 11 shows the accuracy on the test set and the tree size (where applicable) for LMT, C4.5 and SimpleLogistic as a function of the training set size (note the log scale). It can be seen that the accuracy

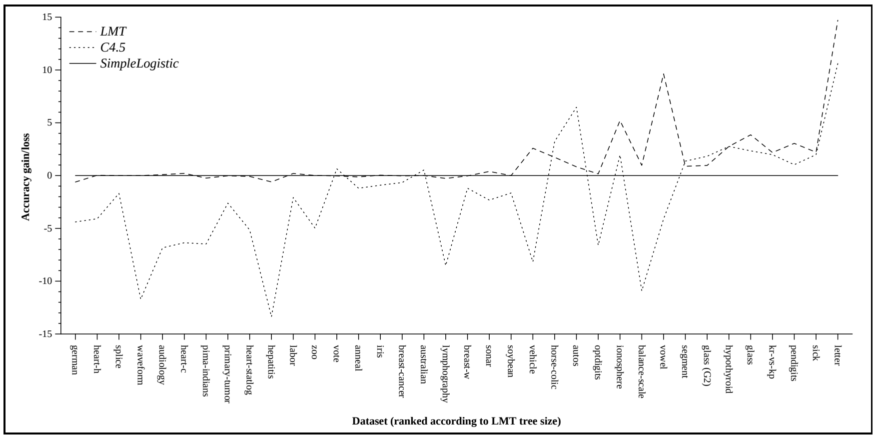

Figure 12. Gain/loss in accuracy of LMT and C4.5 over SimpleLogistic. Datasets are ordered (left to right) by increasing size of LMT tree.

on the test set increases monotonically with the number of training examples for every method. The learning curves of SimpleLogistic and C4.5 cross at around 200 training examples. For small training sets, a bias towards simple models pays off because it allows more stable estimates and does not overfit, while the less biased model space of tree induction allows nonlinear patterns to be captured if enough data is available. More specifically, the learning curve for logistic regression levels off at a training set size of about 100 instances because the method cannot fit the nonlinear part of the underlying distribution, while tree induction continues to improve its estimate with more data.

Looking at LMT, we see that the accuracy is almost identical to that of SimpleLogistic for 25, 50 and 100 instances, but continues to increase for larger datasets. The models built by LMT are more accurate than those of C4.5 even for large training sets. LMT reaches the same accuracy as C4.5 at about half the number of training instances. The graph visualizing the tree sizes shows that the LMT algorithm starts to build a sizeable tree structure around the point the learning curves of SimpleLogistic and C4.5 cross, which indicates that the pruning method correctly detects at what point a tree structure is superior to a linear logistic model. Finally, we note that both tree induction methods build larger and larger trees asymptotically to fit the nonlinear distribution (note the data is noise-free), but the trees built by LMT are significantly smaller than the standard classification trees.

Figure 12 shows a similar study using the UCI datasets. We sorted the datasets according to the size of the trees produced by LMT and then plotted the difference in accuracy of LMT over SimpleLogistic and C4.5 over SimpleLogistic. Although the curves are not as smooth as for the artificial problem, it is clear that LMT is adjusting its tree structure to match the complexity of the different domains. On the left-hand side of the graph, where LMT's models are the smallest, accuracy is equal to SimpleLogistic and generally better than C4.5. The domains on the right-hand side of the graph require more complex models to capture their structure. In all these cases LMT outperforms SimpleLogistic, and in most cases C4.5 as well.

These examples illustrates how the LMT algorithm smoothly scales model complexity from a simple linear model as produced by logistic regression to a more complex combination of tree structure and logistic regression models.

5.4.2. *Comparing LMT to other Enhanced Tree Learners*

This section compares LMT to several other enhanced decision tree learning algorithms. In particular, we present results for NBTree, LTree (using linear discriminants for splitting and at the leaves) and two logistic tree algorithms: LTree (using logistic regression for splitting and at the leaves) and Lotus. When building logistic model trees, Lotus can build the logistic regression functions using either all the numeric attributes present in the data, or by just selecting one of the attributes. These two modes often gave very different results in our experiments. Lotus can only be applied to two class problems. Furthermore, a logistic model tree can only be fit if the data contains at least one numeric attribute. On the heart-h dataset Lotus did not give a reasonable result initially—this was due to an attribute with 99 percent of its values missing. We removed this attribute from the dataset before running Lotus on it. On the labour and horse-colic datasets we were unable to get a result from Lotus using multiple logistic regression at all.

Table VII shows the average classification accuracy of LMT compared to NBTree, LTree using linear discriminants (LTreeLin) and LTree using logistic regression (LTreeLog). Apart from the dataset balance-scale, LMT performs as well or better than the other methods. Compared to NBTree, LMT is significantly more accurate on 10 datasets, with most of those wins applying to the larger datasets. Compared to LTree using linear discriminants, LMT is significantly more accurate on 15 datasets and significantly less accurate on one (the aforementioned balance-scale). When logistic regression is used with LTree, LMT is significantly more accurate on 17 datasets and less accurate on balance-scale. As was the case with NBTree, LMT's superiority occurs more often than not on the larger datasets.

If the significance of the individual differences is not taken into account, Table VIII shows that the win/loss ratio for LMT compared to NBTree is 27/9. This difference is statistically significant according

Table VII. Mean classification accuracy and standard deviation for LMT vs. NBTree, LTree with linear discriminants and LTree with logistic regression.

Dataset	LMT	NBTree		LTreeLin		LTreeLog	
labor	91.37±11.01	91.70±12.32		84.57±14.78		86.47±17.76	
zoo	94.89±6.44	94.95±6.24		93.79±6.67		94.87±6.79	
lymphography	84.10±10.00	80.89±8.77		80.30±8.98		76.90±10.07	•
iris	95.80±4.89	93.53±5.64		98.00±3.35		97.13±4.57	
hepatitis	84.21±8.21	81.36±10.80		84.24±8.85		83.43±9.72	
glass (G2)	77.28±9.90	80.75±9.40		72.39±10.30		72.85±10.42	
autos	76.13±8.84	77.18±9.71		57.74±10.40	•	60.34±10.69	•
sonar	76.32±9.58	77.16±10.46		73.50±9.07		74.25±9.04	
glass	69.15±8.99	70.16±9.67		65.27±10.42		64.77±9.90	
audiology	84.19±7.17	76.82±7.38	•	75.01±7.89	•	75.01±7.89	•
heart-statlog	83.22±6.50	80.59±7.12		83.52±6.28		83.00±6.83	
breast-cancer	74.91±6.29	70.99±7.94		70.58±6.90		70.45±6.78	•
heart-h	84.00±6.33	81.33±6.66		80.10±7.51		82.46±6.54	
heart-c	83.51±6.67	80.60±6.29		80.70±7.80		80.66±7.59	
primary-tumor	47.63±5.84	47.50±6.49		45.29±6.33		45.29±6.33	
ionosphere	92.99±4.13	89.49±5.12		88.95±5.10	•	88.18±5.06	•
horse-colic	83.66±6.13	81.88±6.31		82.11±6.23		81.68±6.50	
vote	95.90±2.67	95.03±3.29		95.72±2.97		95.51±3.20	
balance-scale	89.71±2.68	75.83±5.32	•	92.86±3.22	∘	92.78±3.49	∘
soybean	93.43±2.59	92.87±3.07		90.73±3.04	•	91.16±3.09	•
australian	85.04±3.84	85.07±4.03		84.99±3.91		84.64±4.09	
breast-w	96.18±2.20	96.60±2.04		96.88±1.99		96.75±2.04	
pima-indians	77.08±4.65	75.18±5.05		76.73±4.83		76.64±4.69	
vehicle	83.04±3.70	71.03±4.55	•	79.52±3.81	•	79.32±3.72	•
anneal	99.52±0.73	98.53±1.23	•	95.88±2.36	•	96.56±1.70	•
vowel	93.94±2.55	92.33±3.05		79.70±4.09	•	78.72±4.53	•
german	75.37±3.53	74.07±4.10		74.90±3.47		74.94±3.41	
segment	96.28±1.36	95.22±1.44		96.70±1.27		96.71±1.21	
splice	95.86±1.17	95.43±1.14		92.39±1.63	•	92.85±1.53	•
kr-vs-kp	99.64±0.35	97.81±2.05	•	98.15±0.90	•	98.08±0.93	•
hypothyroid	99.54±0.39	99.58±0.38		98.98±0.52	•	99.25±0.40	•
sick	98.95±0.60	97.82±0.74	•	98.40±0.62	•	98.37±0.65	•
waveform	86.96±1.58	80.12±2.17	•	83.96±1.81	•	84.55±1.67	•
optdigits	97.28±0.64	89.27±1.38	•	96.47±0.75	•	95.94±0.82	•
pendigits	98.56±0.32	95.26±0.69	•	97.09±0.49	•	97.43±0.44	•
letter	92.13±0.74	86.69±0.66	•	88.18±0.68	•	88.07±0.74	•

•, ∘ statistically significant win or loss for LMT

Table VIII. Pairwise comparison of classification accuracy for LMT, NBTree, LTree with linear discriminants and LTree with logistic regression: number indicates how often method in column (significantly) outperforms method in row.

	LMT	NBTree	LTreeLin	LTreeLog
LMT	-	9 (0)	6 (1)	4 (1)
NBTree	27 (10)	-	18 (9)	18 (9)
LTreeLin	30 (15)	18 (6)	-	14 (0)
LTreeLog	32 (17)	18 (5)	20 (0)	-

to a two-tailed sign test: the corresponding p-value is 0.004. Compared to LTree using linear discriminants and LTree using logistic regression, the win/loss ratio is 30/6 and 32/4 respectively. According to a two-tailed sign test there is very strong evidence that LMT is superior to both of these methods (p-values < 0.0001).

Table IX compares the size of the trees produced by LMT with those produced by NBTree and LTree (using both linear discriminants and logistic regression). We can see that LMT never produces significantly larger trees than LTree or NBTree. Compared to NBTree, LMT produces smaller trees on all but five datasets (G2, glass, breast-w, german and splice); compared to LTree with linear discriminants, LMT produces smaller trees in all cases except for horse-colic; and compared with LTree with logistic regression, LMT produces smaller trees on all but five datasets (sonar, ionosphere, horse-colic, breast-w and pima-indians).

Looking at the summary of wins and losses in Table X, and ignoring the significance of the individual differences, we can see that the ratio of wins (smaller trees) to losses (larger trees) for LMT to NBTree is 34/2. Against LTree using linear discriminants it is 35/1 and against LTree using logistic regression it is 34/2. All of these ratios are strongly significant according to a two-tailed sign test (p-values < 0.0001).

Table XI compares LMT with Lotus using simple logistic regression (LotusS) and Lotus using multiple logistic regression (LotusM) on the two-class datasets. Table XII summarizes the results. We can see that LMT is never significantly less accurate than either mode of Lotus and is significantly more accurate than LotusS on six datasets and significantly more accurate than LotusM on five datasets. Ignoring the significance of the individual differences, LMT is more accurate than LotusS on 13 datasets and less accurate on one. This ratio is significant

Table IX. Mean tree size (number of leaves) and standard deviation for LMT vs. NBtree, LTree with linear discriminants and LTree with logistic regression.

Dataset	LMT	NBTree		LTreeLin		LTreeLog	
labor	1.05±0.26	2.81±0.60	•	2.51±0.73	•	2.00±0.00	•
zoo	1.05±0.26	4.82±0.96	•	7.13±0.34	•	7.45±0.50	•
lymphography	1.18±0.80	6.44±2.11	•	2.26±0.48	•	2.66±1.24	•
iris	1.11±0.53	2.60±1.42	•	3.00±0.00	•	3.00±0.00	•
hepatitis	1.05±0.30	6.04±1.81	•	2.17±0.64	•	2.31±0.87	•
glass (G2)	5.05±3.24	2.94±2.18		10.70±2.72	•	10.04±2.64	•
autos	2.89±5.20	18.67±5.33	•	12.57±5.53	•	11.81±4.79	•
sonar	2.12±1.51	7.52±1.23	•	7.02±2.23	•	2.11±0.49	
glass	7.46±3.55	5.12±4.01		21.67±3.01	•	20.83±2.54	•
audiology	1.01±0.10	13.50±3.43	•	26.05±2.19	•	26.05±2.19	•
heart-statlog	1.04±0.32	5.16±2.17	•	3.76±3.07	•	2.69±2.14	•
breast-cancer	1.12±1.20	9.17±9.76	•	3.24±1.27	•	3.26±1.49	•
heart-h	1.00±0.00	7.13±4.69	•	2.91±2.37	•	3.34±2.95	•
heart-c	1.03±0.17	9.42±4.16	•	9.51±4.98	•	10.03±5.19	•
primary-tumor	1.03±0.17	5.15±5.87	•	29.37±6.27	•	29.37±6.27	•
ionosphere	4.40±1.86	8.50±1.71	•	8.73±2.39	•	5.91±2.96	
horse-colic	2.78±2.58	15.92±7.28	•	2.24±1.04		2.16±0.94	
vote	1.08±0.44	9.88±3.05	•	2.27±0.71	•	2.00±0.00	•
balance-scale	4.73±2.61	8.90±5.37	•	14.35±3.39	•	14.03±3.42	•
soybean	2.44±5.44	21.27±11.71	•	32.78±2.21	•	32.48±1.69	•
australian	1.15±0.59	10.88±9.33	•	4.69±4.36	•	4.20±3.99	•
breast-w	1.24±0.99	3.14±2.73		2.04±0.28	•	2.12±0.86	
pima-indians	1.03±0.30	3.04±2.38	•	4.59±4.79	•	3.80±4.45	
vehicle	2.64±1.97	28.40±5.92	•	29.88±8.75	•	25.52±9.37	•
anneal	1.11±0.31	30.00±8.73	•	8.57±2.63	•	15.38±2.73	•
vowel	4.81±1.23	42.02±1.78	•	66.00±5.91	•	60.90±5.18	•
german	1.00±0.00	11.21±15.74		8.64±6.82	•	9.34±7.96	•
segment	4.85±2.05	43.34±6.08	•	28.12±3.10	•	29.74±2.44	•
splice	1.00±0.00	2.39±7.29		38.49±9.18	•	20.99±8.98	•
kr-vs-kp	7.92±0.56	24.59±11.19	•	24.74±3.95	•	24.17±5.63	•
hypothyroid	5.61±0.93	26.58±8.23	•	17.49±2.52	•	14.17±1.81	•
sick	13.56±3.12	52.44±11.25	•	26.30±3.10	•	22.02±3.85	•
waveform	1.00±0.00	47.74±18.34	•	104.74±22.84	•	77.56±27.53	•
optdigits	3.45±1.66	174.70±14.37	•	39.22±4.60	•	49.33±6.38	•
pendigits	11.30±2.78	222.93±20.09	•	119.90±5.67	•	96.36±9.29	•
letter	41.73±8.68	724.17±23.50	•	836.16±17.44	•	784.25±31.49	•

•, ◦ statistically significant smaller or larger trees for LMT

Table X. Pairwise comparison of tree size (number of leaves) for LMT, NBTree, LTree with linear discriminants and LTree with logistic regression: number indicates how often method in column has (significantly) larger tree size than method in row.

	LMT	NBTree	LTreeLin	LTreeLog
LMT	-	34 (31)	35 (35)	34 (31)
NBTree	2 (0)	-	17 (11)	14 (11)
LTreeLin	1 (0)	19 (12)	-	11 (0)
LTreeLog	2 (0)	22 (15)	22 (0)	-

Table XI. Mean classification accuracy and standard deviation for LMT vs Lotus using simple logistic regression (LotusS) and Lotus using multiple logistic regression (LotusM).

Data Set	LMT	LotusS	LotusM
labor	91.37±11.01	74.07±18.18 •	—
hepatitis	84.21±8.21	79.74±9.18	82.08±6.74
glass (G2)	77.28±9.90	75.77±9.97	69.99±10.49
sonar	76.32±9.58	72.38±9.13	72.27±7.92
heart-statlog	83.22±6.50	77.63±7.16 •	83.67±6.43
heart-h	84.00±6.33	80.25±6.42	78.24±6.75 •
heart-c	83.51±6.67	76.63±6.72 •	77.49±7.49 •
ionosphere	92.99±4.13	89.04±4.57 •	87.72±5.57 •
horse-colic	83.66±6.13	83.86±5.67	—
credit-rating	85.04±3.84	84.35±4.18	85.14±4.03
breast-w	96.18±2.20	94.61±2.66 •	96.44±2.13
pima-indians	77.08±4.65	75.08±5.14	77.47±4.39
german-credit	75.37±3.53	72.69±3.11	72.55±3.36 •
sick	98.95±0.60	97.84±0.66 •	96.94±1.00 •

•, ◦ statistically significant win or loss for LMT

at the one percent level according to a two-tailed sign test (p-value of 0.0018). Against LotusM, the ratio of wins to losses is 8/4 in favour of LMT. The p-value from the sign test in this case is 0.388. Consequently, there is only very weak evidence that LMT is superiour to LotusM.

Table XIII gives the average tree size and standard deviation for LMT, LotusS and LotusM. Table XIV summarizes the results. LMT builds significantly smaller trees than LotusS on seven out of the 14

Table XII. Pairwise comparison of classification accuracy for LMT, Lotus using simple logistic regression (LotusS) and Lotus using multiple logistic regression (LotusM): number indicates how often method in column (significantly) outperforms method in row.

	LMT	LotusS	LotusM
LMT	-	1 (0)	4 (0)
LotusS	13 (6)	-	6 (0)
LotusM	8 (5)	6 (0)	-

datasets and significantly larger trees on one. On this single dataset (sick) LMT is more accurate than LotusS, so the extra structure is justified. Against LotusM, LMT produces significantly smaller trees on three of the datasets and significantly larger trees on four. In all of the cases where LMT builds larger trees it is more accurate than LotusM (significantly so on two of the four datasets: ionosphere and sick). Ignoring the significance of the individual differences, LMT builds smaller trees than LotusS on 10 datasets and larger trees on four. Against LotusM the ratio of wins to losses is 4/8 in favour of LotusM. In both these cases there is only weak evidence of the superiority of one method over the other according to a two-tailed sign test (p-values of 0.18 and 0.388 respectively).

5.4.3. *Comparing LMT to Multiple-Tree Models*

This section compares LMT to methods that build models consisting of a set of trees: M5' for classification and boosted C4.5. In order to solve classification problems with M5'—a learner that estimates numeric target variables—we convert the classification problem into a regression problem as described in Section 2.2. The final model consists of a set of model trees, one for every class. When boosting trees using AdaBoost.M1, multiple C4.5 trees are built on reweighted versions of the training data, as explained in Section 3.6. For boosting, it is not clear a priori how many boosting iterations should be performed. AdaBoost.M1 was run with a maximum of 10 and 100 boosting iterations (though boosting can be stopped sooner, if the error of one of the classifiers built on the reweighted training data reaches zero or exceeds 0.5).

Table XIII. Mean tree size (number of leaves) and standard deviation for LMT vs. Lotus using simple logistic regression (LotusS) and Lotus using multiple logistic regression (LotusM).

Data Set	LMT	LotusS	LotusM
labor	1.05±0.26	1.41±0.51	—
hepatitis	1.05±0.30	1.21±0.76	1.00±0.00
glass (G2)	5.05±3.24	2.87±1.40	1.14±0.35 ∘
sonar	2.12±1.51	1.65±0.95	1.00±0.00 ∘
heart-statlog	1.04±0.32	3.57±1.86 •	1.00±0.00
heart-h	1.00±0.00	3.33±1.80 •	1.99±0.98 •
heart-c	1.03±0.17	3.66±1.77 •	1.47±0.63
ionosphere	4.40±1.86	3.59±0.85	1.00±0.00 ∘
horse-colic	2.78±2.58	3.90±2.30	—
australian	1.15±0.59	2.58±0.75 •	2.03±0.30 •
breast-w	1.24±0.99	4.51±1.89 •	1.02±0.14
pima-indians	1.03±0.30	2.86±0.97 •	1.00±0.00
german-credit	1.00±0.00	2.37±0.65 •	2.11±0.40 •
sick	13.56±3.12	8.33±2.73 ∘	2.70±1.19 ∘

•, ∘ statistically significantly smaller or larger trees for LMT

Table XIV. Pairwise comparison of tree size (number of leaves) for LMT, Lotus using simple logistic regression (LotusS) and Lotus using multiple logistic regression (LotusM): number indicates how often method in column has (significantly) larger tree size than method in row.

	LMT	LotusS	LotusM
LMT	-	10 (7)	4 (3)
LotusS	4 (1)	-	0 (0)
LotusM	8 (4)	12 (0)	-

Table XV gives the classification accuracy of LMT, M5' for classification, and boosted C4.5 trees using AdaBoost.M1 with 10 and 100 boosting iterations. Table XVI summarizes the wins and losses. LMT outperforms M5' for classification: it is significantly more accurate on 11 datasets and significantly less accurate on only one. If the significance of the differences is not taken into account, LMT has a win/loss ratio

Table XV. Mean classification accuracy and standard deviation for LMT vs. M5' for classification and boosted C4.5.

Dataset	LMT	M5'		AdaBoost(10)		AdaBoost(100)	
labor	91.37±11.01	85.13±16.33		87.17±14.28		88.73±14.27	
zoo	94.89±6.44	94.48±6.43		96.15±6.11		96.35±6.07	
lymphography	84.10±10.00	80.35±9.32		80.87±8.63		84.72±8.41	
iris	95.80±4.89	94.93±5.62		94.33±5.22		94.53±5.05	
hepatitis	84.21±8.21	82.38±8.79		82.38±8.01		84.93±7.79	
glass (G2)	77.28±9.90	81.08±8.73		85.17±7.75	○	88.72±6.42	○
autos	76.13±8.84	76.03±10.00		85.46±7.23	○	86.77±6.81	○
sonar	76.32±9.58	78.37±8.82		79.22±8.70		85.14±7.84	○
glass	69.15±8.99	71.30±9.08		75.15±7.59	○	78.78±7.80	○
audiology	84.19±7.17	76.83±8.62	•	84.84±7.46		84.70±7.57	
heart-statlog	83.22±6.50	82.15±6.77		78.59±7.15		80.44±7.08	
breast-cancer	74.91±6.29	70.40±6.84	•	66.75±7.61	•	66.36±8.18	•
heart-h	84.00±6.33	82.44±6.39		78.68±7.43		78.42±7.20	•
heart-c	83.51±6.67	82.14±6.65		78.76±7.09	•	80.00±6.55	
primary-tumor	47.63±5.84	45.26±6.22		41.65±6.55	•	41.65±6.55	•
ionosphere	92.99±4.13	89.92±4.18	•	93.05±3.92		94.02±3.83	
horse-colic	83.66±6.13	83.23±5.40		81.63±6.19		81.85±5.70	
vote	95.90±2.67	95.61±2.77		95.51±3.05		95.26±3.13	
balance-scale	89.71±2.68	87.76±2.23		78.35±3.78	•	76.11±4.09	•
soybean	93.43±2.59	92.90±2.61		92.83±2.85		93.32±2.81	
australian	85.04±3.84	85.39±3.87		84.01±4.36		86.43±3.98	
breast-w	96.18±2.20	95.85±2.15		96.08±2.16		96.70±2.18	
pima-indians	77.08±4.65	76.56±4.71		71.81±4.85	•	73.89±4.75	
vehicle	83.04±3.70	78.66±4.38	•	75.59±3.99	•	77.87±3.58	•
anneal	99.52±0.73	98.64±1.13		99.59±0.70		99.63±0.65	
vowel	93.94±2.55	80.93±4.68	•	92.89±2.82		96.81±1.93	○
german	75.37±3.53	74.99±3.31		70.91±3.60	•	74.53±3.26	
segment	96.28±1.36	97.31±1.05	○	98.12±0.92	○	98.61±0.69	○
splice	95.86±1.17	95.40±1.17		94.58±1.16	•	94.95±1.25	•
kr-vs-kp	99.64±0.35	99.21±0.50	•	99.59±0.31		99.62±0.30	
hypothyroid	99.54±0.39	99.44±0.38		99.65±0.31		99.69±0.31	
sick	98.95±0.60	98.41±0.62	•	98.99±0.50		99.05±0.50	
waveform	86.96±1.58	82.51±1.60	•	81.32±1.90	•	85.06±1.71	•
optdigits	97.28±0.64	95.43±0.93	•	97.38±0.62		98.51±0.47	○
pendigits	98.56±0.32	97.87±0.47	•	98.99±0.31	○	99.41±0.26	○
letter	92.13±0.74	91.05±0.62	•	95.53±0.43	○	97.25±0.37	○

•, ○ statistically significant win or loss for LMT

Table XVI. Pairwise comparison of classification accuracy for LMT, M5' for classification and boosted C4.5: number indicates how often method in column (significantly) outperforms method in row.

	LMT	M5'	AdaBoost(10)	AdaBoost(100)
LMT	-	5 (1)	14 (6)	19 (9)
M5'	31 (11)	-	19 (11)	23 (15)
AdaBoost(10)	22 (9)	17 (4)	-	30 (7)
AdaBoost(100)	17 (7)	13 (1)	5 (1)	-

of 31/5 against M5'. According to a two-tailed sign test there is very strong evidence that LMT is superior to M5' for classification (p-value < 0.0001). Comparing LMT with boosted C4.5 trees, it can be seen that performing 100 boosting iterations is superior to performing only 10. We conclude that performing 10 boosting iterations is not enough and concentrate on AdaBoost(100).

Looking at the number of significant wins and losses, we can see that LMT is comparable to AdaBoost(100) (with seven wins and nine losses). However, the relative performance of the two schemes really depends on the datasets. Interestingly, there are several datasets where boosting achieves a similar gain in accuracy compared to simple tree induction as LMT, i.e. there was a win for LMT against C4.5 and there is neither a loss nor a win of LMT against AdaBoost(100). This is the case for seven datasets. On the nine datasets where boosted trees outperform LMT, they achieve a higher accuracy than any other scheme, and the gain is quite impressive (up to seven percentage points higher than the next-best classifier). It is clear that the extra structure induced by boosting is justified on these datasets.

The seven datasets for which AdaBoost(100) is significantly less accurate than LMT can be split into two groups. For two datasets (breast-cancer, heart-h), boosting seems to have failed: there was no win of LMT over C4.5, but there is one over AdaBoost(100). It is reasonable to expect that using a more advanced boosting scheme (for example, controlling the number of boosting iterations by cross-validation) would make these losses disappear. For the other five datasets, boosting seems to have no impact on performance (balance-scale, primary-tumor) or increases performance compared to C4.5, but not as much as building logistic model trees (waveform, vehicle, splice).

Applying a two-tailed sign test to the win/loss ratios (ignoring the significance of individual differences) for LMT against AdaBoost(10) and AdaBoost(100) shows that there is only weak evidence to suggest the superiority of one method over the other. Against AdaBoost(10) the win/loss ratio is 22/14 (p-value of 0.243) and against AdaBoost(100) the win/loss ratio is 17/19 (p-value of 0.868).

6. Conclusions

In this work we have introduced a new method for learning logistic model trees, called LMT, that builds on earlier work on model trees. This method employs the LogitBoost algorithm (Friedman et al., 2000) for building the logistic regression functions at the nodes of a tree and uses the well-known CART algorithm for pruning. We have shown how LogitBoost can be used to select the most relevant attributes in the data when performing logistic regression by performing a simple regression in each iteration and stopping before convergence to the maximum likelihood solution. The optimal number of iterations is determined by cross-validation. Our experiments show that, when used as a standalone learning algorithm, this method yields final models that contain significantly fewer parameters than the ones generated by standard maximum likelihood logistic regression while never significantly decreasing accuracy and sometimes significantly increasing it. Another benefit obtained from using LogitBoost for building the logistic regression functions in a model tree is that a separate smoothing process is not required. This is because they can be constructed by incrementally refining logistic regression functions fit at higher levels in the tree.

Our experiments demonstrate that LMT produces models that are often more accurate than those produced by C4.5, CART and standalone logistic regression on real-world datasets. LMT also outperforms well-known enhanced decision tree learners such as LTree and NBTree and the logistic model tree algorithm Lotus. More surprisingly, LMT appears to be competitive with boosted C4.5 trees.

LMT produces a single tree containing binary splits on numeric attributes, multiway splits on nominal ones, and logistic regression models at the leaves, and the algorithm ensures that only relevant attributes are included in the latter. The result is not quite as easy to interpret as a standard decision tree, but much more intelligible than a committee of multiple trees or more opaque classifiers like kernel-based estimators. Like other tree induction methods, LMT can be used 'off the shelf'—it does not require any tuning of parameters by the user.

6.1. Future Work

There are several issues that provide directions for future work. Probably the most important drawback of logistic model tree induction is the high computational complexity compared to simple tree induction. Although the asymptotic complexity of LMT is acceptable compared to other methods (see Section 4.3), the algorithm is quite slow in practice. Most of the time is spent fitting the logistic regression models at the nodes with the LogitBoost algorithm. It would be worthwhile looking for a faster way of fitting the logistic models that achieves the same performance (including variable selection). The heuristic discussed in Section 4.3.1 significantly speeds up the algorithm without decreasing prediction accuracy, but it is admittedly ad-hoc and not very intuitive. Further research might yield a more principled way of determining the optimum number of LogitBoost iterations to be performed at a node without an additional cross-validation.

A further issue is the handling of missing values. At present, LMT uses a simple global imputation scheme for filling in missing values. Although our experiments do not suggest a particular weakness of the algorithm on datasets with missing values, a more sophisticated scheme for handling them might improve accuracy for domains where missing values occur frequently.

References

Blake, C. and C. Merz: 1998, 'UCI Repository of machine learning databases'. [www.ics.uci.edu/~mlearn/MLRepository.html].

Breiman, L., H. Friedman, J. A. Olshen, and C. J. Stone: 1984, *Classification and Regression Trees*. Wadsworth.

Chan, K. Y. and W. Y. Loh: 2004, 'LOTUS: An Algorithm for Building Accurate and Comprehensible Logistic Regression Trees'. *Journal of Computational and Graphical Statistics* **13**(4).

Frank, E., Y. Wang, S. Inglis, G. Holmes, and I. H. Witten: 1998, 'Using Model Trees for Classification'. *Machine Learning* **32**(1), 63–76.

Freund, Y. and R. E. Schapire: 1996, 'Experiments with a New Boosting Algorithm'. In: *Proc International Conference on Machine Learning*. pp. 148–156, Morgan Kaufmann.

Friedman, J., T. Hastie, and R. Tibshirani: 2000, 'Additive Logistic Regression: a Statistical View of Boosting'. *The Annals of Statistic* **38**(2), 337–374.

Gama, J.: 2004, 'Functional Trees'. *Machine Learning* **55**(3), 219–250.

Hastie, T., R. Tibshirani, and J. Friedman: 2001, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag.

Ihaka, R. and R. Gentleman: 1996, 'R: A Language for Data Analysis and Graphics'. *Journal of Computational and Graphical Statistics* **5**(3), 299–314.

Kohavi, R.: 1996, 'Scaling Up the Accuracy of Naive Bayes Classifiers: A Decision-Tree Hybrid'. In: *Proc 2nd International Conference on Knowledge Discovery and Data Mining*. Menlo Park, CA, pp. 202–207, AAAI Press.

Landwehr, N., M. Hall, and E. Frank: 2003, 'Logistic Model Trees'. In: *Proc 14th European Conference on Machine Learning*. pp. 241–252, Springer-Verlag.

Lim, T.-S., W. Loh, and Y. Shih: 2000, 'A Comparison of Prediction Accuracy, Complexity, and Training Time for Thirty-Three Old and New Classification Algorithms'. *Machine Learning* **40**(3), 641–666.

Lubinsky, D.: 1994, 'Tree Structured Interpretable Regression'. In: D. Fisher and H. Lenz (eds.): *Learning from Data*. pp. 387–398, Springer-Verlag.

Malerba, D., A. Appice, M. Ceci, and M. Monopoli: 2002, 'Trading-Off Local versus Global effects of Regression Nodes in Model Trees'. In: M.-S. Hacid, Z. W. Ras, D. A. Zighed, and Y. Kodratoff (eds.): *ISMIS 2002*. pp. 393–402, Springer-Verlag.

Nadeau, C. and Y. Bengio: 2003, 'Inference for the Generalization Error'. *Mach. Learn.* **52**(3), 239–281.

Perlich, C., F. Provost, and J. Simonoff: 2003, 'Tree Inductions vs. Logistic Regression: A Learning-curve Analysis'. *Journal of Machine Learning Research* **4**, 211–255.

Quinlan, J. R.: 1992, 'Learning with Continuous Classes'. In: *Proc 5th Australian Joint Conference on Artificial Intelligence*. pp. 343–348, World Scientific Publishing Company Incorporated.

Quinlan, R.: 1993, *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Wang, Y. and I. Witten: 1997, 'Inducing model trees for continuous classes'. In: *Proc of Poster Papers, European Conf. on Machine Learning*. Prague, Czech Republic.

Witten, I. H. and E. Frank: 2000, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implemenations*. San Francisco: Morgan Kaufmann.